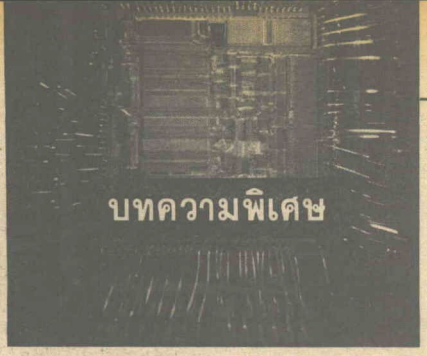


บทความพิเศษ

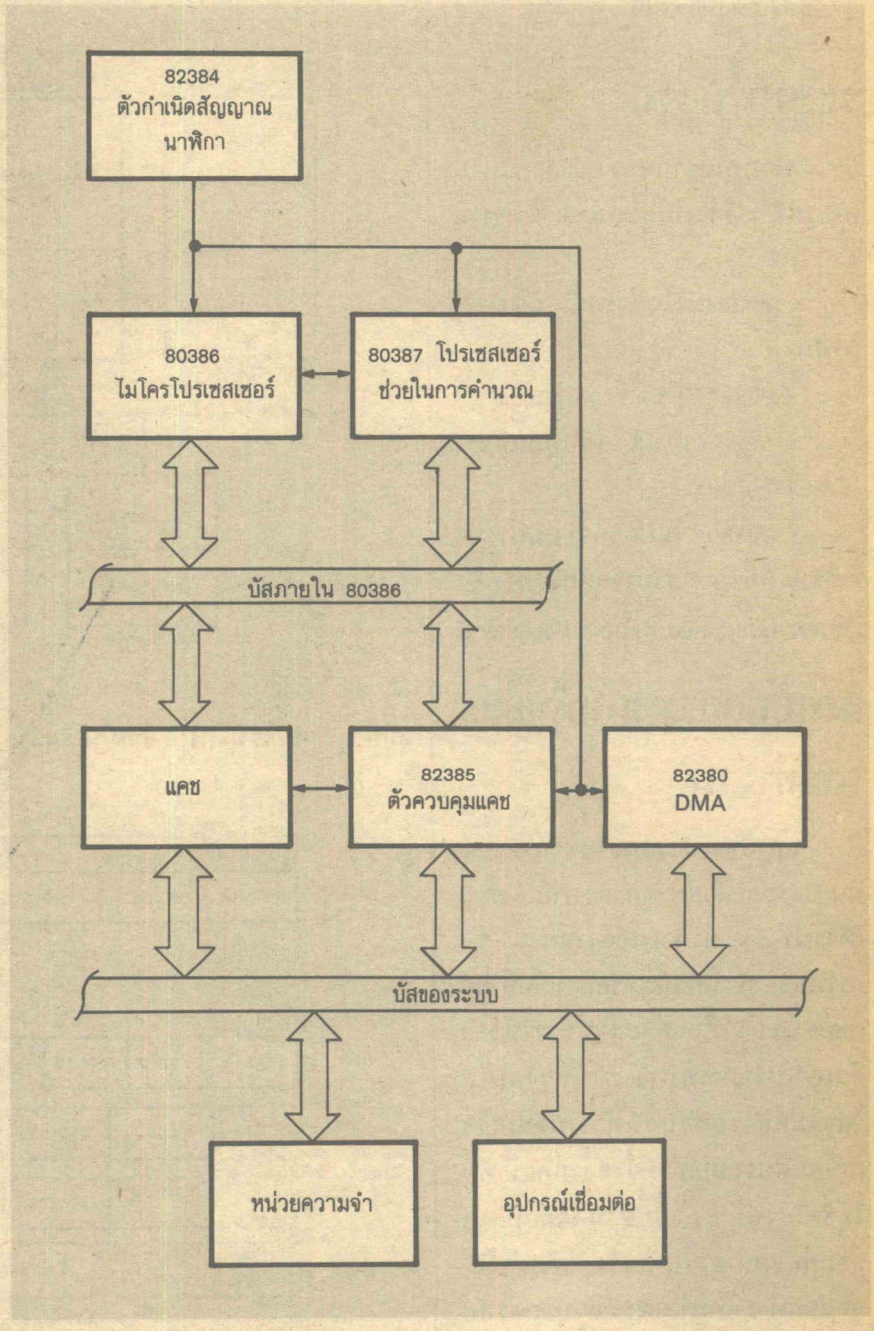
เริ่มต้นกับ 80386

เอนก ศิรจริยาวิภิวัฒน์*

ถวัลย์ ตั้งลิตานนท์*

เชื่อว่าพวกเราคงคุ้นกับซีพียู 8088 และ 8086 กันมานานแล้ว ตอนนี้ก็ได้ข่าวว่าอินเทลจะออกซีพียู 80486 ซึ่งทั้งราคาและความสามารถเหลือเกินจริง ๆ ก่อนจะถึง 80486 ขอให้เรามาติดตามเทคโนโลยีของซีพียู 80386 ของบริษัทอินเทลที่พบในเครื่องไมโครระดับ 80386 เครื่อง PS/2 โมเดล 70 และ 80 ของไอบีเอ็ม และขณะนี้ก็มีใช้ในเครื่อง AT กันมาก ทั้งนี้เพราะพวกเราจะมีโอกาสได้สัมผัสกับมันแน่ในอนาคตอันใกล้นี้ ตอนที่ผู้เขียนเริ่มสัมผัสกับเทคโนโลยีทางด้านคอมพิวเตอร์ใหม่ ๆ นั้น ผู้เขียนก็เริ่มกับเครื่องแอปเปิ้ลทู ซึ่งตอนนั้นคิดว่าหน่วยความจำขนาด 64 กิโลไบต์ของแอปเปิ้ลมันเพียงพอจนมองไม่ออกว่า การ์ด 128 กิโลไบต์ที่มีอยู่จะใช้ประโยชน์กับมันได้อย่างมีประสิทธิภาพอย่างไร แต่ครั้นผู้เขียนเริ่มจับไอบีเอ็มพีซี รู้จักระบบที่ใหญ่ขึ้น รู้จักเครื่องเมนเฟรม เริ่มจับงานด้าน LAN, CAD/CAM, การจำลองแบบ ก็เริ่มรู้สึกว่าเครื่องไอบีเอ็มพีซี/เอ็กซ์ที หรือแม้แต่ไอบีเอ็มพีซี/เอทีที่มีอยู่มันช้าไปถนัดใจจนเมื่อได้มาใช้เครื่อง 80386 เพื่องานระบบมัลติยูสเซอร์, มัลติทาสกิ้ง, LAN จึงพบว่าซีพียูตัวนี้เหมาะสมที่จะใช้กับงานเหล่านี้มาก ในบทความนี้จะกล่าวถึง 80386 อย่างกว้าง ๆ ซึ่งรายละเอียดเราคงได้มีโอกาสนำเสนอท่านในโอกาสต่อไป

80386 เป็นซีพียูขนาด 32 บิต สามารถต่อหน่วยความจำ (physical memory) ได้สูงสุด 4 กิกะไบต์ ผลิตด้วยเทคโนโลยี CHMOS II ประกอบด้วยทรานซิสเตอร์ภายใน 275,000 ตัว มีทั้งหมด 132 ขา มีขาแอดเดรส และ



รูปที่ 1 ระบบพื้นฐานของ 80386

*ผู้ช่วยนักวิจัยโครงการวิจัย และพัฒนาเครื่องต้นแบบคอมพิวเตอร์ 32 บิต

ขาดาต้า แยกกันต่างหาก สำหรับตัวที่วิ่งที่สัญญาณนาฬิกา 20 เมกะเฮิรตซ์ สามารถปฏิบัติคำสั่งได้มากกว่า 5 ล้านคำสั่งต่อวินาที ซึ่งความเร็วขนาดนี้เทียบเท่าเครื่องมินิคอมพิวเตอร์ได้อย่างสบาย ในแต่ละงาน (task) บนเครื่อง 80386 นั้น สามารถใช้เซกเมนต์ได้ 16,381 เซกเมนต์ แต่ละเซกเมนต์ใช้หน่วยความจำได้สูงสุด 4 กิกะไบต์ (คงไม่ต้องบ่นกันแล้วซินะว่าเซกเมนต์มันเล็กเกินไป) ซึ่งทำให้ 80386 อ้างหน่วยความจำได้สูงสุดถึง 64 เทราไบต์

ระบบพื้นฐาน

ระบบพื้นฐานของ 80386 เป็นไปตามรูปที่ 1 ซึ่งจะเห็นชิพช่วยการทำงานดังนี้

- 82384 เป็นตัวกำเนิดสัญญาณนาฬิกา
- 80387 เป็นตัวช่วยคำนวณ
- 82385 เป็นตัวควบคุมแคช (CACHE)
- 82380 เป็นตัวควบคุมการทำดีเอ็มเอ ที่มีความสามารถหลายทาง จึงเรียกว่า Integrated System Peripheral

สถาปัตยกรรมภายในและการทำงาน

สถาปัตยกรรมภายในของ 80386 นั้น ประกอบด้วยส่วนการทำงาน 6 ส่วน (ดังรูปที่ 2) ที่ทำงานอย่างขนาน ซึ่งการทำงานขนานกันนี้เราเรียกว่า ไปป์ไลน์ (pipe line) คือแต่ละส่วนทำงานของตัวเองเสร็จก็เริ่มทำงานในช่วงต่อไปโดยไม่ต้องรอส่วนอื่นทำงานจนเสร็จเหมือนกับระบบเก่า ซึ่งความคิดนี้เห็นได้ชัดในรูปที่ 3 เมื่อส่วนติดต่อบัสนำข้อมูลส่วนที่ 1 เข้ามาเสร็จ ก็เริ่มนำข้อมูลส่วนที่ 2 เข้ามา ขณะที่ส่วนถอดรหัสเริ่มตีความหมายของข้อมูลส่วนที่ 1 และเป็นเช่นนี้เรื่อยไปกับส่วนอื่น ๆ สำหรับส่วนการทำงาน 6 ส่วน ก็ได้แก่

1. ส่วนติดต่อบัส

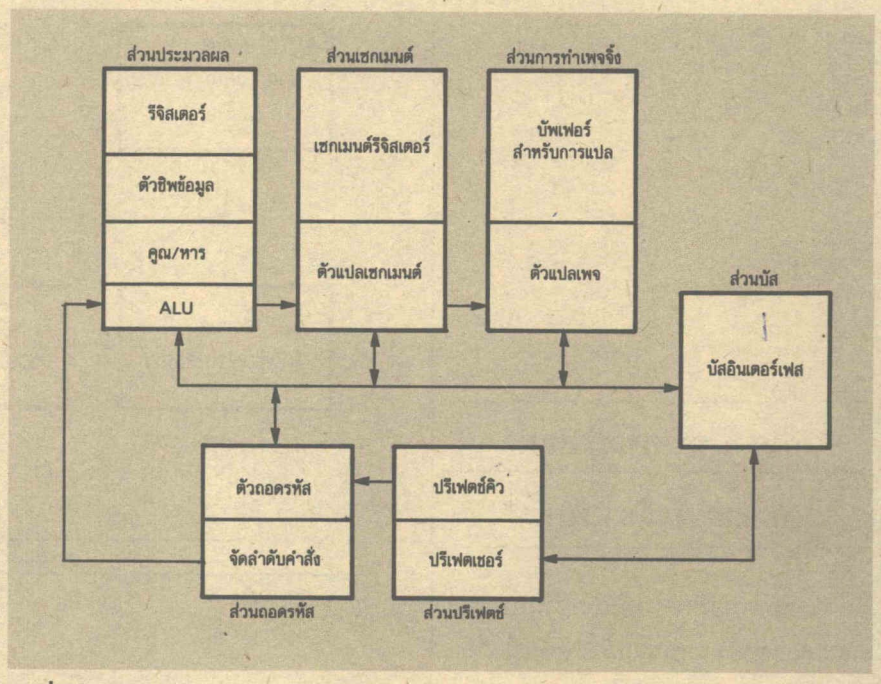
เป็นตัวเชื่อมต่อระหว่าง 80386 กับอุปกรณ์รอบ ๆ ตัว 80386 เป็นตัวรับส่งข้อมูล และทางผ่านของคำสั่ง (code fetches) เป็นตัวให้สัญญาณ และประมวลผลสัญญาณของบัสไซเคิล ซึ่งมีแอดเดรส, ดาต้า, การเข้าถึงหน่วยความจำ, อินพุต/เอาต์พุต และการใช้โปรเซสเซอร์ร่วม (coprocessor)

2. ส่วนการเฟตช์คำสั่ง

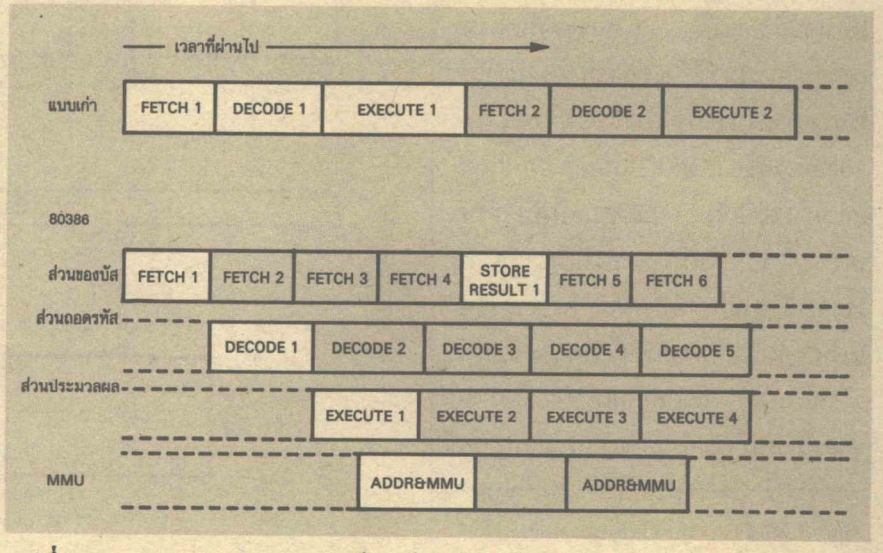
จะดึงคำสั่ง (fetch) เข้ามาโดยผ่านทางส่วนติดต่อบัส จะเก็บเป็นคิวขนาด 16 ไบต์ เพื่อรอการประมวลผลโดยส่วนถอดรหัสต่อไป

3. ส่วนถอดรหัส

เป็นตัวแปลคำสั่งที่ได้จากคิวของส่วนการเฟตช์คำสั่งไปเป็นไมโครโค้ด การถอดรหัสคำสั่งนี้ที่ได้จะเก็บไว้ในคิวขนาด 3 คำสั่ง เพื่อรอการประมวลผลของส่วนประมวลผลต่อไป



รูปที่ 2 ส่วนการทำงาน 6 ส่วนใน 80386



รูปที่ 3 แสดงการทำงานแบบไปป์ไลน์

4. ส่วนประมวลผล

เป็นตัวประมวลคำสั่งประกอบด้วยส่วนย่อยอีก 3 ส่วนคือ

- ส่วนควบคุม

จะมีไมโครโค้ด และฮาร์ดแวร์พิเศษที่เป็นแบบขนาน เพื่อความรวดเร็วในการคูณ หาร และการคำนวณแอดเดรสยังผล (effective address : EA)

- ส่วนของข้อมูล

จะมี ALU, รีจิสเตอร์ใช้งานทั่วไปขนาด 32 บิต อยู่ 8 ตัว, มีตัวชิฟข้อมูลขนาด 64 บิต ซึ่งสามารถชิฟได้ทีละหลาย ๆ บิตใน 1 สัญญาณนาฬิกา และเป็นตัวจัดข้อมูลให้กับส่วนควบคุม

- ส่วนการทดสอบการป้องกัน

เป็นตัวเช็คเซกเมนต์ว่าเกิดการผิดพลาดหรือไม่

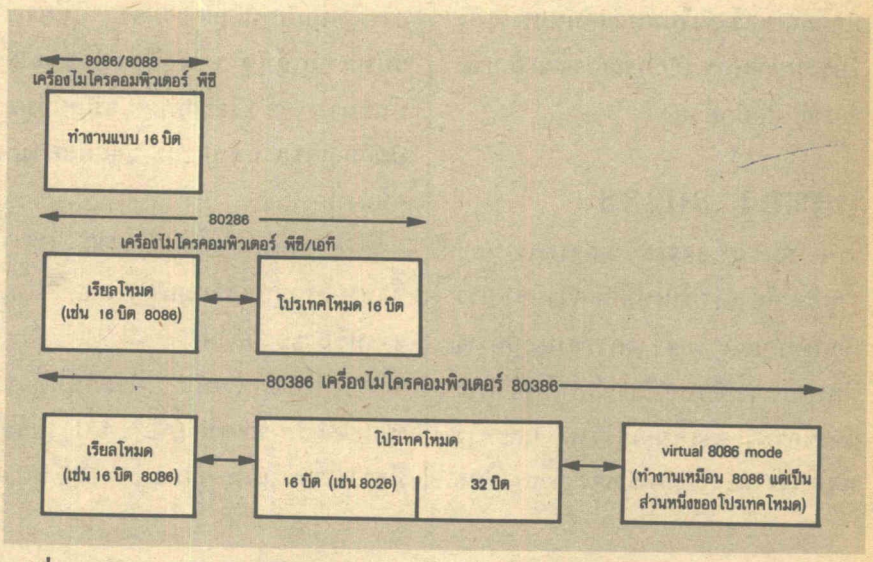
5. ส่วนเซกเมนต์

เป็นตัวแปลลอจิคัลแอดเดรส ให้เป็นแอดเดรสเชิงเส้นให้กับหน่วยประมวลผล จะมี Segment Descripter Cache เก็บเซกเมนต์ที่กำลังใช้งานอยู่เพื่อความรวดเร็วในการแปล

6. ส่วนเพจจิ้ง

เมื่อใช้ 80386 โดยมีเพจจิ้ง (paging) ส่วนนี้ จะแปลแอดเดรสเชิงเส้นที่ได้จากส่วนเซกเมนต์หรือส่วนเฟตช์คำสั่งให้เป็นฟิสิคัลแอดเดรส ถ้าไม่มีการทำเพจจิ้ง ก็จะไม่มีการแปลเกิดขึ้น เช่นเดียวกับส่วนเซกเมนต์ ส่วนนี้จะมี Page Decripter Cache เพื่อความรวดเร็วในการแปล ฟิสิคัลแอดเดรสที่ได้นี้ จะถูกส่งให้ส่วนติดต่อบัสต่อไป

สำหรับการทำงานของ 80386 จะแบ่งเป็นโหมดการทำงาน 3 โหมด (ดังรูปที่ 4) คือ



รูปที่ 4 โหมดการทำงานต่าง ๆ ของ 80386

1. เรียลโหมด (Real Mode)

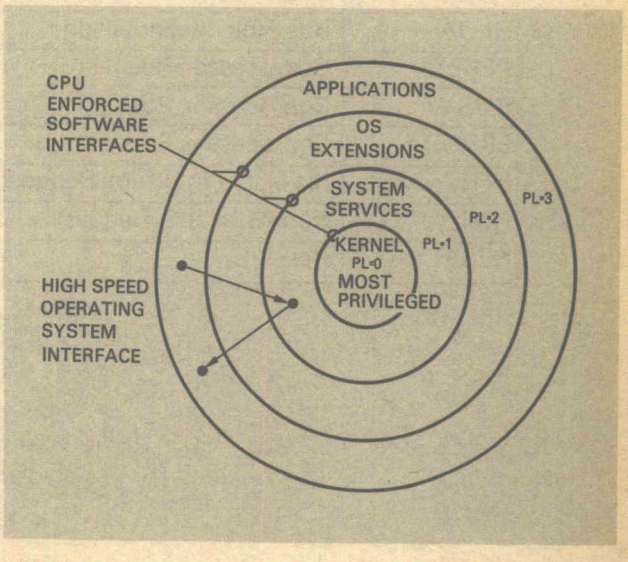
สำหรับโหมดนี้ จะเป็นโหมดที่เข้ากันได้กับซอฟต์แวร์ของ 8086 ที่มีอยู่เดิมสามารถอ้างแอดเดรสได้สูงสุด 1 เมกะไบต์ และมีขนาดเซกเมนต์ได้แค่ 64 กิโลไบต์ แต่สามารถใช้รีจิสเตอร์ขนาด 32 บิต และคำสั่งของ 80386 เกือบทั้งหมดได้

2. โปรเทคโหมด (Protected Mode)

การทำงานในโหมดนี้ จะใช้ความสามารถของ 80386 ได้เต็มความสามารถ มีการทำงานแบบเพจจิ้ง สามารถอ้างแอดเดรสได้สูงสุด 64 เทราไบต์ สำหรับโหมดที่ทำงานเป็น 80286 จะอ้างแอดเดรสได้สูงสุด 16 เมกะไบต์ จะมีระบบการป้องกัน 4 ชั้น(ดังรูปที่ 5) สนับสนุนการทำงานแบบมัลติทาสกิ้ง เพื่อแยกและป้องกันโปรแกรมของผู้ใช้จากโปรแกรมอื่นและระบบจัดการ (Operating System: OS)

3. เวอร์ชวล 8086 โหมด (Virtual 8086 Mode)

เป็นโหมดของ 8086 ที่ถูกสร้างในโปรเทคโหมด ซึ่งจะทำให้เรารู้สึกว่ามี 8086 ได้หลาย ๆ ตัวพร้อมกัน โดยใช้ความสามารถของกลไกการป้องกัน (protected machanism) เช่น ขณะที่คนหนึ่งทำงานสเปรดชีต อีกคนเล่นเอ็มเอสดอส และขณะเดียวกันอีกคนเล่นยูนิกซ์อยู่ ซึ่งนั่นก็คือการทำงานแบบ



รูปที่ 5 ระบบการป้องกัน 4 ชั้น

มัลติยูสเซอร์ ยังไม่หมดแค่นั้นเพราะยังรับระบบจัดการ (OS) ของ 80286 หรือ 80386 ได้อีกด้วย

บัสของ 80386

บัสของ 80386 มีการทำงานอยู่ 7 ชนิด คือ การอ่านหน่วยความจำ การเขียนหน่วยความจำ การอ่านอุปกรณ์อินพุต การเขียนอุปกรณ์เอาต์พุต การเฟตช์คำสั่ง การอินเตอร์รัพต์ และการหยุดทำงาน (Halt/Shutdown) โดยการติดต่อกับหน่วยความจำ อุปกรณ์อินพุต เอาต์พุต หรืออุปกรณ์อื่นจะใช้บัสขนาน ซึ่งประกอบด้วยบัสข้อมูล บัสแอดเดรส บัสสถานะ และบัสควบคุมดังนี้

- ขาดาต้า 32 ขา (D0 - D31) ซึ่งสามารถรับส่งข้อมูลขนาด 8, 16, 24 หรือ 32 บิตได้

- แอดเดรสบัส ประกอบด้วยขาแอดเดรส 30 ขา (A2 - A31) และมีขาไบต์อนาเบิ้ล อีก 4 ขา (BE#0 - BE#3) ซึ่งขาไบต์อนาเบิ้ลจะสัมพันธ์กับขนาดของข้อมูลที่จะใช้ว่าเป็น 8, 16, 24 หรือ 32 บิต

- บัสสถานะ มีอยู่ 5 ชนิด คือ

1. สถานะของแอดเดรส (ADS#) เป็นตัวบอกว่าอยู่ในช่วงการใช้แอดเดรสบัส

2. การอ่านและเขียน (W/R#) เป็นตัวบอกว่าอยู่ในช่วงการอ่านหรือเขียนข้อมูล

3. การติดต่อหน่วยความจำกับ

Signal Name	Signal Function	Active State	Input/ Output	Input Synch or Asynch to CLK 2	Output High Impedence During HLDA?
CLK2	Clock	-	I	-	-
D0-D31	Data Bus	High	I/O	S	Yes
BE0#-BE3#	Byte Enables	Low	O	-	Yes
A2-A31	Address Bus	High	O	-	Yes
W/R#	Write-Read Indication	High	O	-	Yes
D/C#	Data-Control Indication	High	O	-	Yes
M/IO#	Memory-I/O Indication	High	O	-	Yes
LOCK#	Bus Lock Indication	Low	O	-	Yes
ADS#	Address Status	Low	O	-	Yes
NA#	Next Address Request	Low	I	S	-
BS16#	Bus Size 16	Low	I	S	-
READY#	Transfer Acknowledge	Low	I	S	-
HOLD	Bus Hold Request	High	I	S	-
HLDA	Bus Hold Acknowledge	High	O	-	No
PEREQ	Coprocessor Request	High	I	A	-
BUSY#	Coprocessor Busy	Low	I	A	-
ERROR#	Coprocessor Error	Low	I	A	-
INTR	Maskable Interrupt Request	High	I	A-	
NMI	Non-Maskable Request	High	I	A	-
RESET	Reset	High	I	S	-

ตารางที่ 1 บัสของ 80386

การติดต่ออุปกรณ์อินพุต/เอาต์พุต (M/IO#) เป็นตัวบอกว่าติดต่อหน่วยความจำหรือติดต่อกับอุปกรณ์อินพุต/เอาต์พุต

4. ข้อมูลหรือการควบคุม (D/C#) เป็นตัวบอกว่าอยู่ในช่วงการรับส่งข้อมูลหรือช่วงการควบคุม

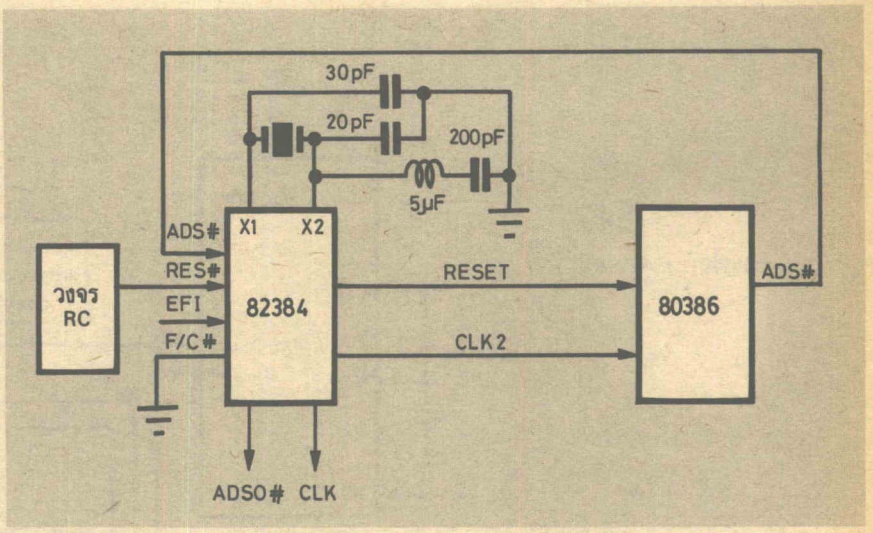
5. ล็อค (LOCK#) เป็นตัวบอกสถานะของการล็อคบัส

- บัสควบคุม เป็นขาอินพุตเพื่อให้วงจรลอจิกภายนอกควบคุมบัสไซเคิล ซึ่งมี

1. เรียบร้อย (Ready#) เป็นตัวบอกการจบบัสไซเคิล

2. แอดเดรสต่อไป (NA#) เป็นขาที่อุปกรณ์บอกว่ายอมให้ใช้แอดเดรสต่อไป ในการทำไปป์ไลน์ เพื่อใช้กับอุปกรณ์ที่ช้าได้

3. บัสขนาด 16 บิต (BS16#) เป็นตัวบอกการใช้ดาต้าบัสแบบ 16 บิต โดยใช้ขา D0-D15 ดังในตารางที่ 1



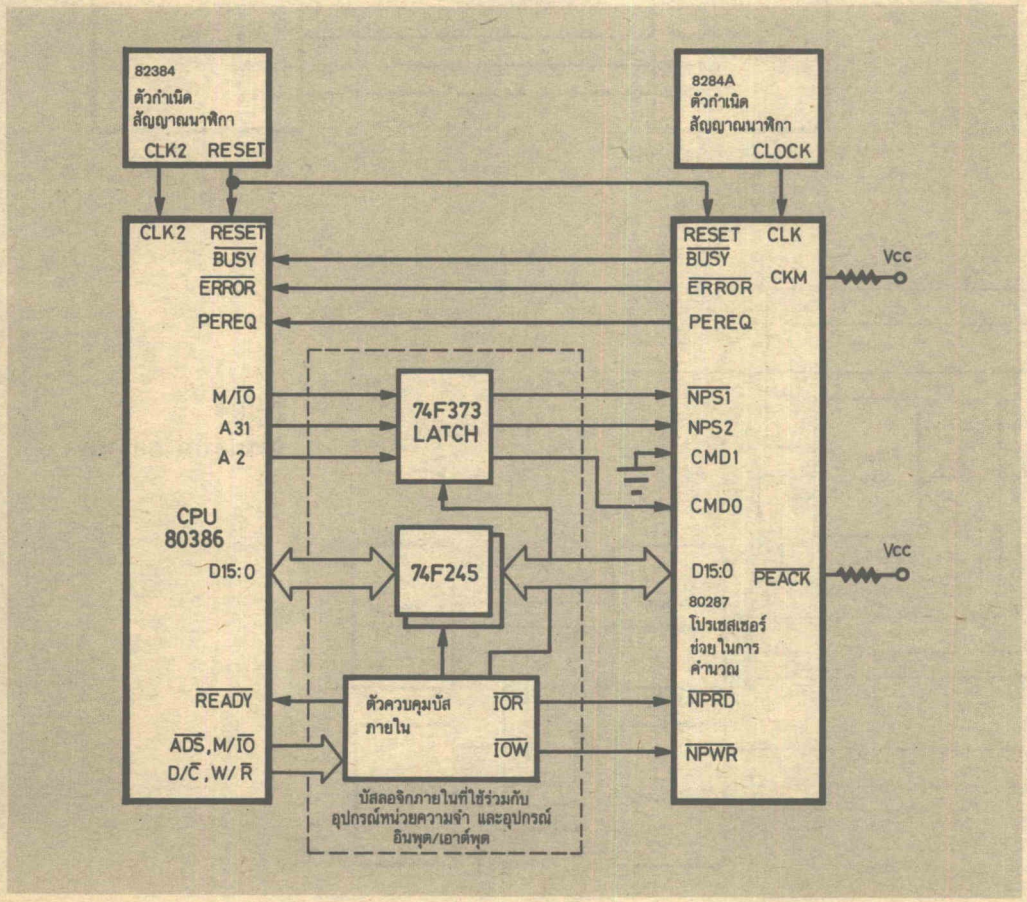
รูปที่ 6 การต่อสัญญาณนาฬิกา

นอกจากที่กล่าวมายังมีขาที่ใช้ควบคุมการทำงานของ 80386 และเชื่อมต่ออุปกรณ์ภายนอกอีก คือ

1. CLK2 เป็นขาที่ใช้ป้อนสัญญาณนาฬิกา จะเป็นการป้อนสัญญาณนาฬิกาความถี่ 2 เท่าของความถี่ที่ 80386 ต้องการ เช่น 80386 ขนาด 20 เมกะเฮิรตซ์ จะต้องใช้สัญญาณนาฬิกาขนาด 40 เมกะเฮิรตซ์ โดยมีวงจรหารความถี่อยู่ภายในตัว 80386 ดังรูปที่ 6

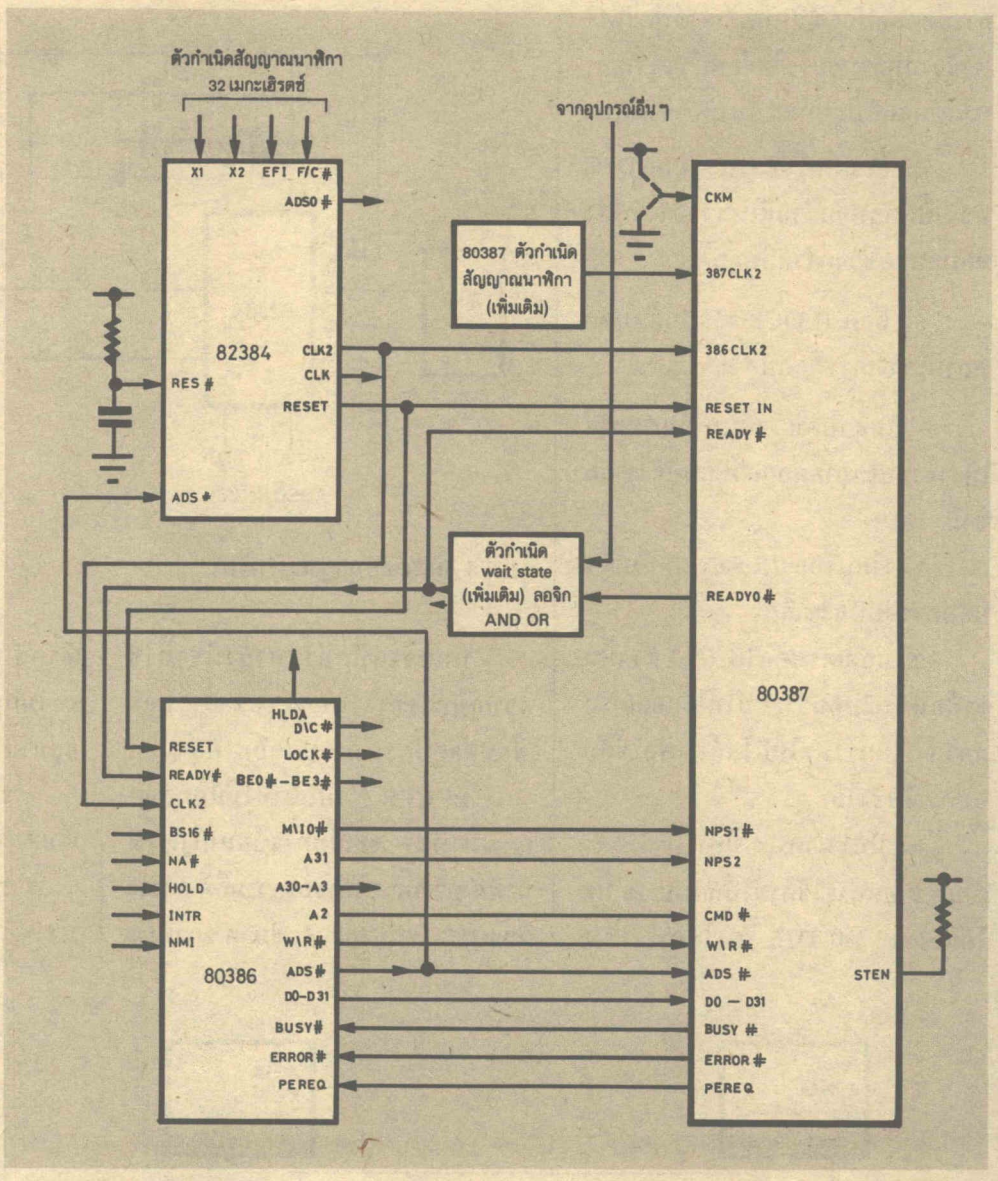
2. RESET เป็นขาที่ทำให้ 80386 เริ่มต้นทำงานใหม่หมด (initialize)

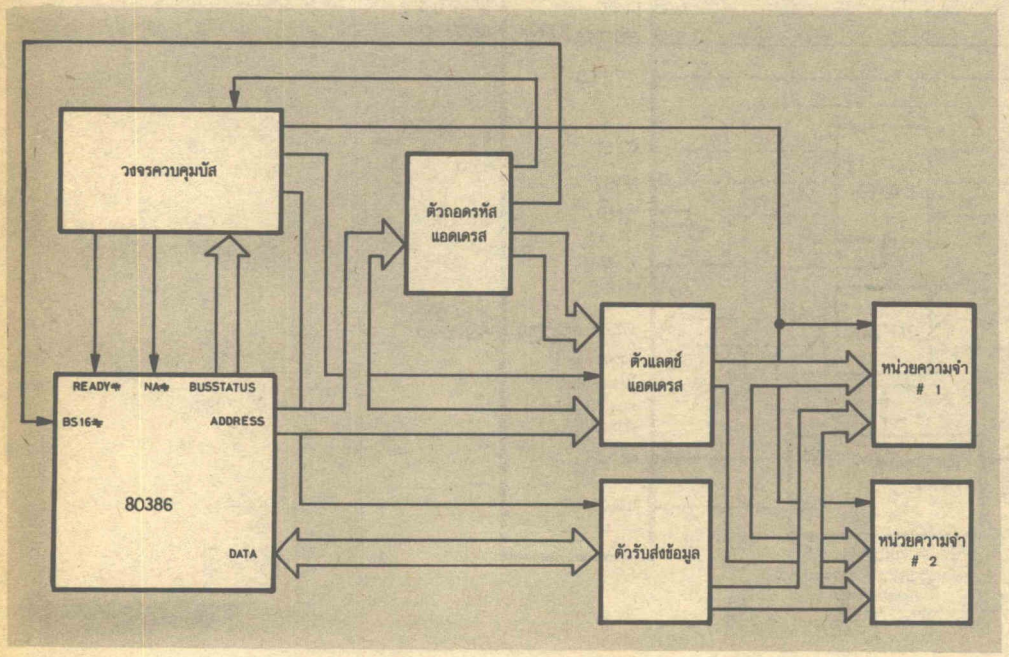
3. HOLD เป็นขาที่อุปกรณ์ภายนอกต้องการใช้บัสของ 80386 โดย



รูปที่ 7 การต่อกับ 80287

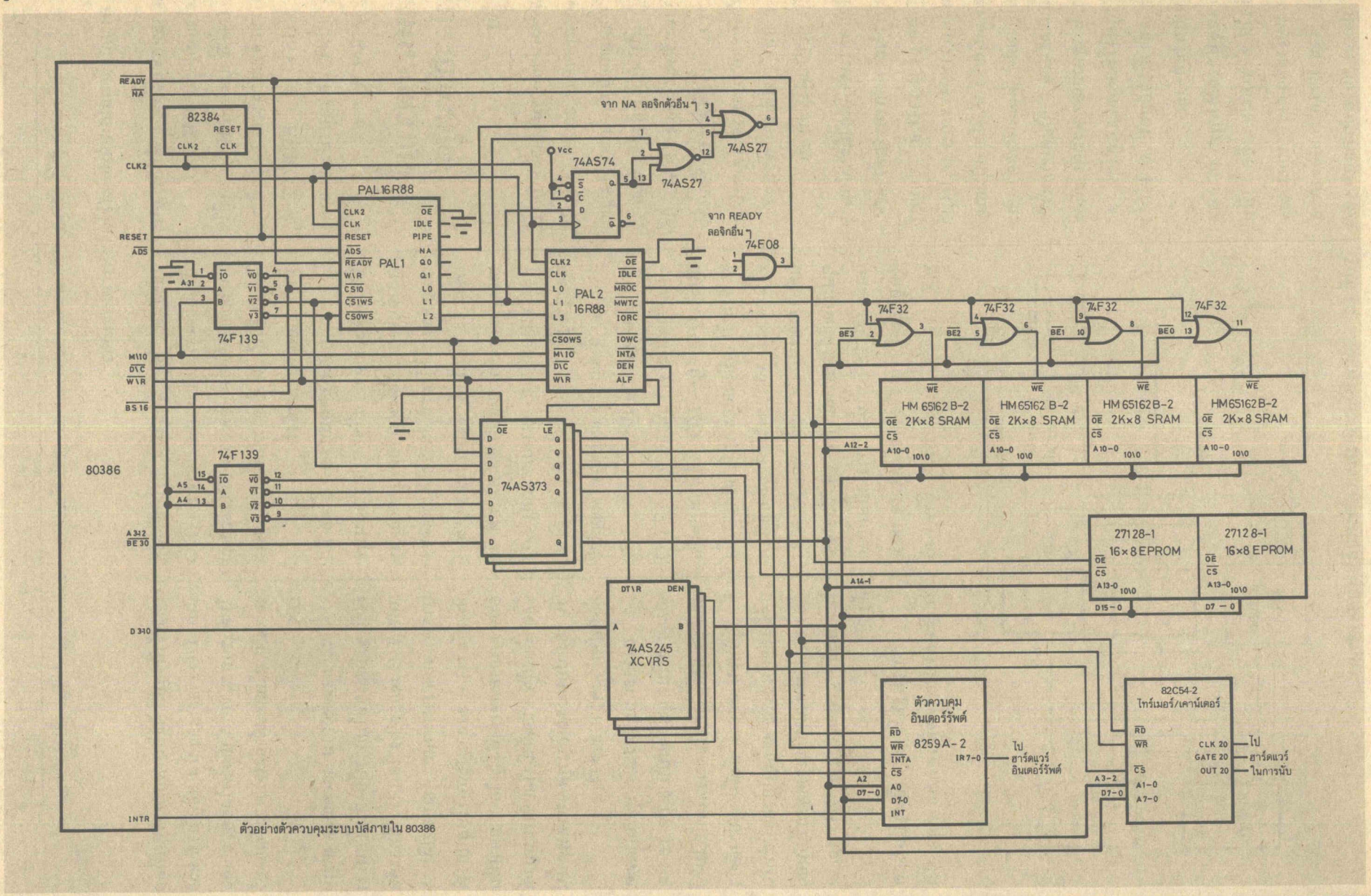
รูปที่ 8
การต่อกับ 80387



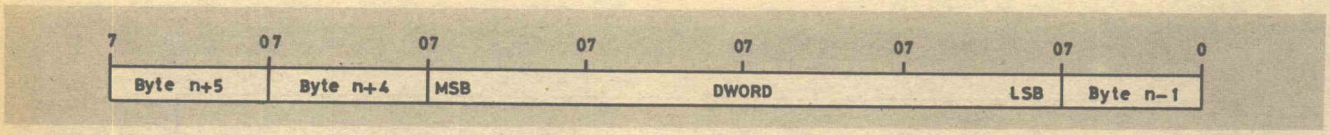


รูปที่ 9
การต่อหน่วยความจำ

รูปที่ 10 วงจรควบคุมบัส



ตัวอย่างตัวควบคุมระบบบัสภายใน 80386



รูปที่ 11 การเก็บข้อมูลแบบดับเบิลเวิร์ดในหน่วยความจำ

80386 จะตอบสนองโดยการให้สัญญาณ Hold Acknowledge (HLDA)

4. INTR และ NMI เป็นขาที่ใช้สำหรับการขออินเตอร์รัพต์ ดังรูปที่ 7,8

5. BUSY #, ERROR #, PEREQ เป็นขาที่ต่อกับโปรเซสเซอร์ร่วม 80287 และ 80387

การต่อหน่วยความจำและอุปกรณ์อินพุต/เอาต์พุต

การต่อหน่วยความจำและอุปกรณ์อินพุต/เอาต์พุตของ 80386 นิยมใช้ PAL (Programable Array Logic) ซึ่งจะสามารถโปรแกรมฟังก์ชันของการทำงานทางลอจิกลงไปบน PAL ได้ และจะใช้ร่วมกับไอซีตระกูลทีทีแอล ซึ่งบล็อคไดอะแกรมของการต่อหน่วยความจำจะเป็นดังรูปที่ 9 ส่วนวงจรควบคุมบัส จะเป็นดังรูปที่ 10

หน่วยความจำที่ต่อได้จะมีขนาดสูงสุด 4 กิกะไบต์ แต่สามารถอ้างแอดเดรสได้สูงสุด 64 เทราไบต์ ข้อมูลที่เก็บในหน่วยความจำจะเก็บในรูปไบต์เวิร์ด หรือดับเบิลเวิร์ดก็ได้ ถ้าเป็นแบบเวิร์ด จะเก็บไบต์ต่ำที่แอดเดรสต่ำ และไบต์สูงที่แอดเดรสสูง ถ้าเป็นดับเบิลเวิร์ดก็จะเก็บไบต์ต่ำสุดที่แอดเดรสต่ำ และเก็บไบต์สูงสุดไว้ที่แอดเดรสสูงสุด ดังรูปที่ 11 การอ้างถึงข้อมูลนั้นจะใช้อ้างแอดเดรสที่เก็บไบต์ต่ำสุด

การต่ออุปกรณ์อินพุต/เอาต์พุต สามารถต่อได้ 2 ลักษณะ คือ เป็นลักษณะของหน่วยความจำ (memory mapping) ซึ่งสามารถอ้างแอดเดรส (memory space) ของอุปกรณ์อินพุต/เอาต์พุต ได้ 4 กิกะไบต์ หรือการต่อเป็นแอดเดรสแบบอินพุต/เอาต์พุต (I/O mapping) ซึ่งอ้างแอดเดรส (I/O space) ได้ 64 กิโลไบต์ ซึ่งการต่อทั้ง 2 ลักษณะมีความแตกต่างกันดังนี้

1. การเลือกอุปกรณ์ ถ้าต่อเป็น I/O mapping การเลือกอุปกรณ์อินพุต/เอาต์พุต จะทำได้ง่ายกว่าแบบ memory mapping ทั้งนี้เพราะมีการอ้างแอดเดรสเพียง 64 กิโลไบต์ ซึ่งถ้าเป็นแบบ memory mapping จะเป็น 4 กิกะไบต์ ซึ่งต้องใช้สายแอดเดรสมากกว่าแบบ I/O mapping

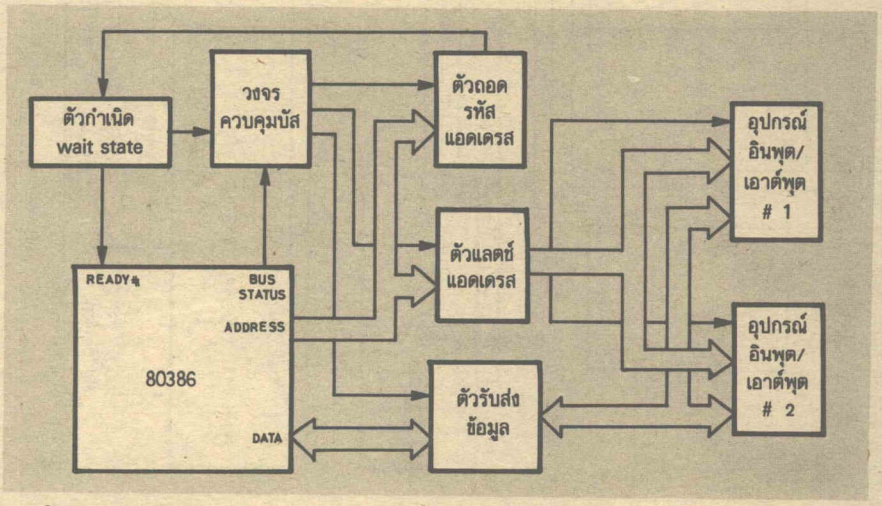
2. ถ้าต่อแบบ memory mapping การเข้าถึงอุปกรณ์นั้นจะใช้คำสั่ง I/O - to - memory, memory - to - I/O และ I/O - to - I/O ได้ดีพอ ๆ กับคำสั่งเปรียบเทียบและคำสั่งทดสอบ ซึ่งเป็นการใช้คำสั่งอย่างมีประสิทธิภาพ แต่ถ้าต่อแบบ I/O mapping จะเข้าถึงอุปกรณ์ได้โดยคำสั่ง IN, OUT, INS และ OUTS เท่านั้น การเคลื่อนย้ายข้อมูลจะทำผ่านรีจิสเตอร์ AL (8 บิต), AX (16 บิต) และ EAX (32 บิต)

การเข้าถึงอุปกรณ์อินพุต/เอาต์พุตที่ต่อแบบ I/O mapping นั้น แอดเดรส 256 ไบต์แรกสามารถเข้าถึงได้โดยตรง และใน 64 กิโลไบต์จะเข้าถึงโดยผ่านรีจิสเตอร์ DX

3. การต่อแบบ memory mapping จะยืดหยุ่นในการป้องกันมากกว่าแบบ I/O mapping โดยการใช้การจัดการหน่วยความจำและคุณสมบัติการป้องกันระบบ จึงสามารถทำให้อุปกรณ์อินพุต/เอาต์พุตเข้าถึงไม่ได้ มองเห็นแต่ถูกป้องกัน หรือเข้าถึงได้อย่างสมบูรณ์ขึ้นอยู่กับอุปกรณ์อินพุต/เอาต์พุตนั้นต่อกับหน่วยความจำช่วงใด

การต่อกับอุปกรณ์อินพุต/เอาต์พุต 80386 สนับสนุนทั้งแบบ 8 บิต 16 บิต และ 32 บิต จะเป็นขนาดใดนั้นไม่ได้ขึ้นอยู่กับขนาดของดาต้าเพียงอย่างเดียว แต่ขึ้นอยู่กับสัญญาณที่อุปกรณ์เหล่านั้นต้องการ และตำแหน่งแอดเดรสที่มันอยู่ใน memory space หรือ I/O space ด้วย

สำหรับรูปที่ 12 เป็นรูปบล็อคไดอะแกรมการต่ออินพุต/เอาต์พุต ขั้น



รูปที่ 12 บล็อคไดอะแกรมของการต่ออินพุต/เอาต์พุต

พื้นฐาน ซึ่งจะมีอุปกรณ์ที่เราเห็นบ่อย ๆ ก็คือ

8259A - ตัวควบคุมอินเตอร์รัพต์

8237 - ตัวควบคุมดีเอ็มเอ

82258 - ตัวควบคุมดีเอ็มเอชั้นสูง

8253, 8254 - ไทเมอร์ที่โปรแกรมได้

8272 - ตัวควบคุมดิสค์ไดรฟ์แบบฟลอปปี้ดิสค์

82062, 82064 - ตัวควบคุมดิสค์-ไดรฟ์แบบฮาร์ดดิสค์

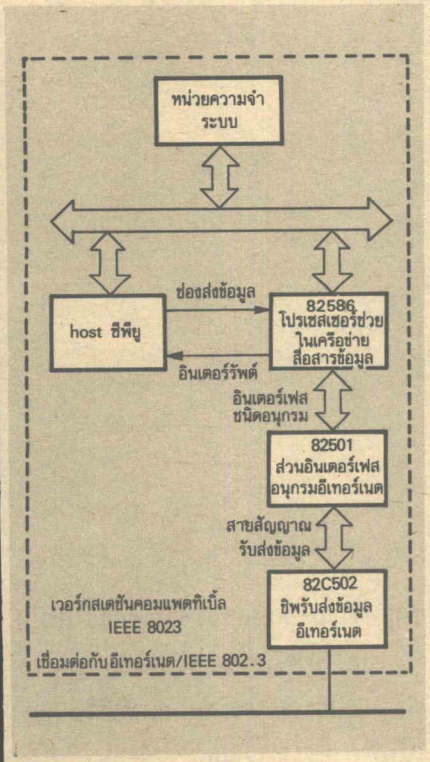
8274-ตัวควบคุมอุปกรณ์สื่อสารข้อมูลแบบอนุกรม หลายโปรโตคอล

8255 - ตัวติดต่ออุปกรณ์ภายนอกที่โปรแกรมได้

8041, 8042 - ตัวติดต่ออุปกรณ์ภายนอกที่สามารถใช้ได้ทั่ว ๆ ไป (Universal)

82380 - ตัวควบคุมการทำดีเอ็มเอที่มีความสามารถหลายทาง

82586 - ตัวประมวลผล LAN



รูปที่ 13 บล็อคไดอะแกรมของการต่อ LAN

ชนิดของข้อมูล

80386 สนับสนุนชนิดของข้อมูลที่ใช้ในภาษาขั้นสูง ซึ่งได้แก่

1. Bit เป็นข้อมูลขนาด 1 บิต

2. Bit Field เป็นกลุ่มของบิตข้อมูล มีขนาดสูงสุด 32 บิต

3. Bit String เป็นกลุ่มของบิตข้อมูล มีขนาดสูงสุด 4 กิกะไบต์

4. Byte เป็นข้อมูลขนาด 8 บิต รวมเครื่องหมาย

5. Unsigned Byte เป็นข้อมูลขนาด 8 บิต ไม่รวมเครื่องหมาย

6. Integer (word) เป็นข้อมูลขนาด 16 บิต รวมเครื่องหมาย

7. Long Integer (Double word) เป็นข้อมูลขนาด 32 บิต รวมเครื่องหมาย

8. Unsigned Integer (word) เป็นข้อมูลขนาด 16 บิต ไม่รวมเครื่องหมาย

9. Unsigned Long Integer (Double word) เป็นข้อมูลขนาด 32 บิต ไม่รวมเครื่องหมาย

10. Signed Quad Word เป็นข้อมูลขนาด 64 บิต รวมเครื่องหมาย

11. Unsigned Quad Word เป็นข้อมูลขนาด 64 บิต ไม่รวมเครื่องหมาย

12. Offset เป็นข้อมูลขนาด 16 บิต หรือ 32 บิต

13. Pointer จะประกอบไปด้วยตัวเลือกเซกเมนต์ขนาด 16 บิต และออฟเซตขนาด 16 บิต หรือ 32 บิต

14. Char เป็นข้อมูลแอสกี

15. String เป็นข้อมูลแบบสตริงขนาด 1 ไบต์ถึง 4 กิกะไบต์ 16.BCD จะเป็นไบต์ (unpack) ของข้อมูล แสดงเลข 0-9 เพียงตัวเดียว

17. Packed BCD จะเป็นไบต์ (pack) ของข้อมูลตัวเลข 0 - 9 อยู่ในแต่ละ 4 บิต อย่างละตัว (1 ไบต์ = 2 ตัวเลข)

18. Floating Point ข้อมูลแบบจำนวนจริง (real number) ขนาด 32, 64 และ 80 บิต ซึ่งจะมีได้เมื่อใช้โปรเซสเซอร์ร่วม (80387)

รีจิสเตอร์

มีอยู่ 31 ตัว ซึ่งแบ่งเป็น

- รีจิสเตอร์สำหรับผู้ใช้ (User Registers) ประกอบด้วย

1. รีจิสเตอร์ทั่วไป (General registers) 8 ตัว

2. รีจิสเตอร์ชี้คำสั่ง (Instruction register) 1 ตัว

3. รีจิสเตอร์แฟล็ก (Flag register) 1 ตัว

4. เซกเมนต์รีจิสเตอร์ (Segment register) 6 ตัว

- รีจิสเตอร์ของระบบ (System Registers) ประกอบด้วย

1. รีจิสเตอร์ควบคุม (Control registers) 3 ตัว

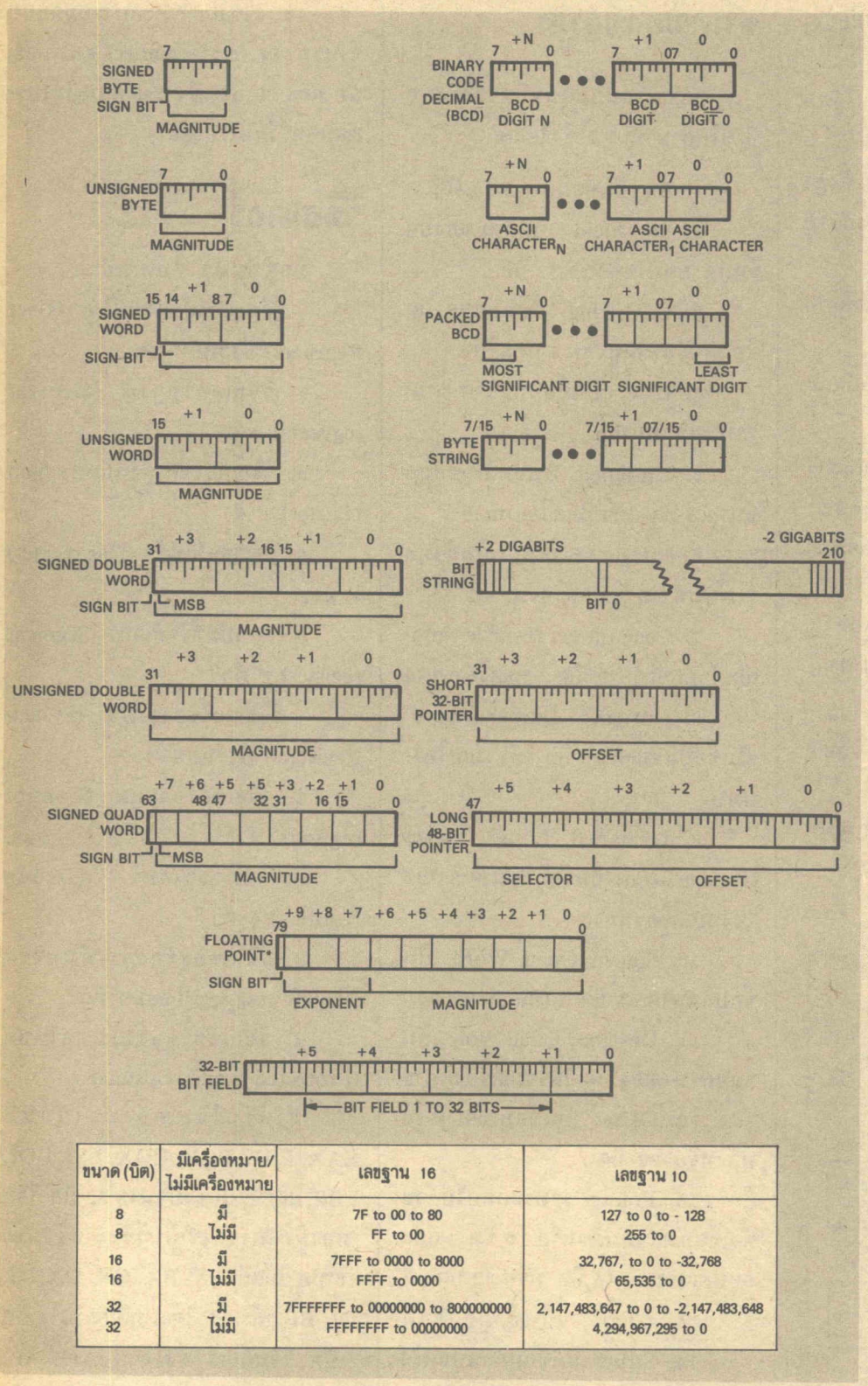
2. ดีบักรีจิสเตอร์ (Debug registers) 6 ตัว

3. รีจิสเตอร์ของการทำเพจจิ้ง (Paging test registers) 2 ตัว

4. รีจิสเตอร์ของโปรเทคโหมด (Protected mode registers) 4 ตัว

สำหรับรีจิสเตอร์ทั่วไปนั้นมี EAX, EBX, ECX, EDX, ESI, EDI, EBP, และ ESP ซึ่งมีขนาด 32 บิต รีจิสเหล่านี้สามารถใช้เป็นรีจิสเตอร์ขนาด 16 บิต ได้แก่ AX, BX, CX, DX, DI, DI, BP และ SP ได้เช่นเดียวกับใช้เป็น 8 บิต รีจิสเตอร์ ซึ่งได้แก่ AH, AL, BH, BL, CH, CL, DH, และ DL ซึ่งใช้กับข้อมูลขนาด 1, 8, 16, 32 และ 64 บิต หรือ bit field ขนาด 1 บิต จนถึง 32 บิต

รีจิสเตอร์ชี้คำสั่ง คือ EIP ซึ่งจะใช้ในลักษณะ 16 บิตก็ได้ซึ่งคือ IP นั่นเอง



รูปที่ 14 ชนิดของข้อมูลที่สนับสนุนโดย 80386

ขนาด (บิต)	มีเครื่องหมาย/ไม่มีเครื่องหมาย	เลขฐาน 16	เลขฐาน 10
8	มี	7F to 00 to 80	127 to 0 to - 128
8	ไม่มี	FF to 00	255 to 0
16	มี	7FFF to 0000 to 8000	32,767, to 0 to -32,768
16	ไม่มี	FFFF to 0000	65,535 to 0
32	มี	7FFFFFFF to 00000000 to 800000000	2,147,483,647 to 0 to -2,147,483,648
32	ไม่มี	FFFFFFFF to 00000000	4,294,967,295 to 0

รีจิสเตอร์แฟล็ก จะมีลักษณะดังรูปที่ 16 บิตที่เป็นเลขศูนย์ คือ บิตที่อินเทลเรียกว่า Undefined Bits ซึ่งบิตเหล่านี้ อินเทลขอสงวนไว้ ถ้าเกิดเราใช้บิตเหล่านี้ทางอินเทลบอกว่า ซอฟต์แวร์ที่เราสร้างขึ้นอาจใช้ไม่ได้หรือเข้ากันไม่ได้กับไมโครโปรเซสเซอร์ ตัวต่อไป ทางที่ดีเราก็ไม่ควรใช้บิตเหล่านี้ดีกว่า ส่วนบิตที่เหลือมีดังนี้

1. VM (Virtual 8086 Mode) บิต 17 ของ EFLAGE เป็นบิตที่ถูกเซต (set) ได้เพียงในโปรเทคโหมดเพื่อเปลี่ยนการทำงานไปเป็นเวอร์ชวล 8086 โหมด

2. RF (Resume Flag) บิต 16 ของ EFLAGE ใช้ร่วมกับดีบักรีจิสเตอร์เพื่อทำ break point

3. NT (Nested Task) บิต 14 ของ EFLAGE เป็นตัวบอกว่างาน

(task) ที่ทำอยู่เป็นโครงข่าย (Nested) ของงาน (task) อื่น ซึ่งมีใช้ในโปรเทคโหมด

4. IOPL (Input/Output Previlege level) บิต 12 และ 13 เพื่อบอกจำนวนสูงสุดของค่า CPL (Current priviledge level) เมื่อใช้คำสั่งเกี่ยวกับอินพุต/เอาต์พุต มีใช้โปรเทคโหมด

5. OF (Overflow flag) บิต 11 ของ EFLAG เพื่อบอกการทำงานที่เกินการกระทำทางบิต ที่มีอยู่ เช่น OF จะถูกเซต (set) เมื่อเป็น 8/16/32 บิต การทำงานเกิดการโอเวอร์โฟลว์ที่บิต 7/15/31 ตามลำดับ

6. DF (Direction Flag) บิต 10 ของ EFLAG เป็นตัวบอกว่า ESI หรือ EDI จะเพิ่มค่าหรือลดค่าเมื่อทำการประมวลผลคำสั่งเกี่ยวกับสตริง ESI หรือ EDI จะลดลงเมื่อบิตนี้ถูกเซต

7. IF (INTR Enable flag) บิต 9 ของ EFLAG ตัวบอกว่าจะยอมรับอินเตอร์รัพต์แบบ Maskable interrupt หรือไม่ จะยอมรับการขออินเตอร์รัพต์เมื่อบิตนี้ถูกเซต

8. TF (Trap Enable Flag) บิต 8 ของ EFLAG ใช้ในการทำ single step (เมื่อถูกเซต) หรือการทำ break point โดยใช้ดีบักรีจิสเตอร์

9. SF (Sign Flag) บิต 7 ของ EFLAG จะถูกเซต (set) เมื่อบิตสูงสุดของผล (result) ถูกเซต

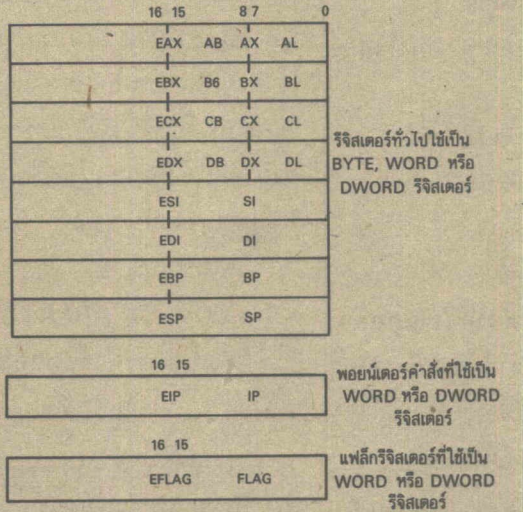
10. ZF (Zero flag) บิต 6 ของ EFLAG จะถูกเซต (set) เมื่อทุกบิตของผล (result) เป็นศูนย์

11. AF (Auxiliary Carry Flag) บิต 4 ของ EFLAG จะใช้ในการบวกลบเลข packed BCD

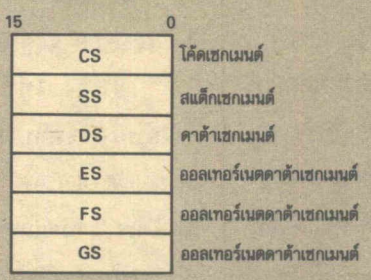
12. PF (Parity Flag) บิต 2 ของ EFLAG จะถูกเซตเมื่อ 8 บิตล่าง (low-order eight bits) มีจำนวนเลข 1 เป็นเลขคู่ (even parity)

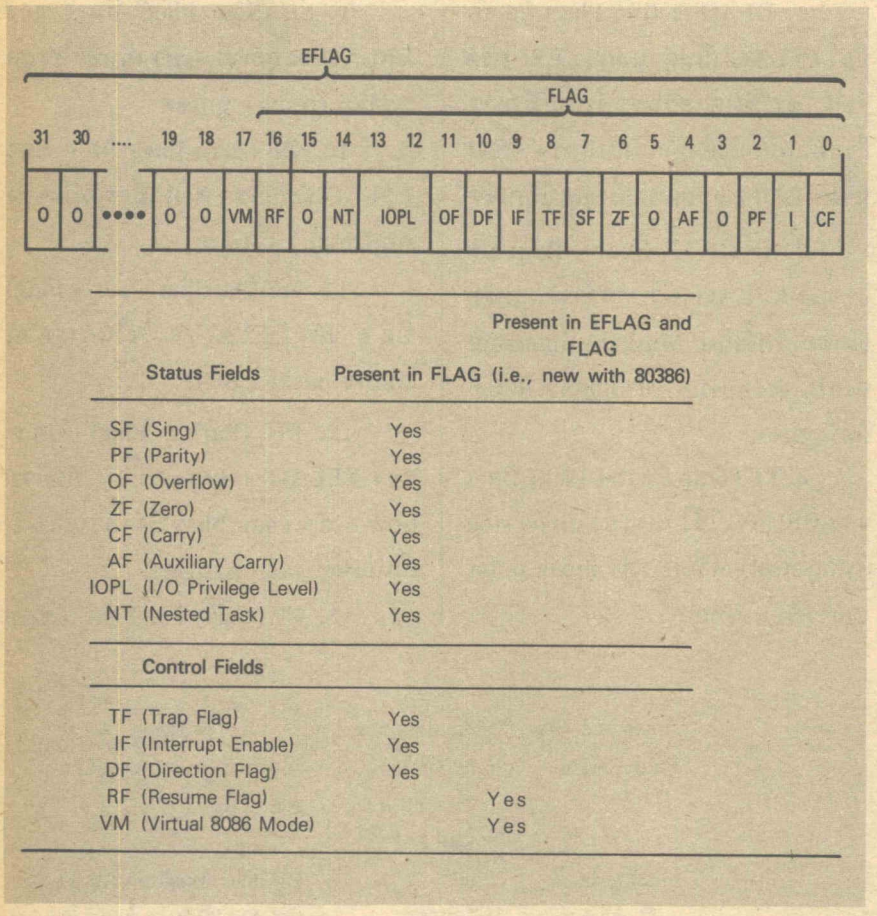
13. CF (Carry Flag) บิต 0 ของ

รูปที่ 15
รีจิสเตอร์
สำหรับผู้ใช้



80386 รีจิสเตอร์	หมายเลขและชื่อ	หน้าที่
รีจิสเตอร์ทั่วไป	8:EAX,EBX,ECX, EDX,ESI,EDI, EBP,ESP	ใช้สำหรับเก็บแอดเดรสและดาต้าทั่วไป
พอยน์เตอร์คำสั่ง	1:EIP	EIP ใช้ชี้คำสั่งต่อไปที่จะปฏิบัติ
แฟล็กรีจิสเตอร์	1:EFLAG	EFLAG ใช้เก็บฟิลด์สถานะ 8 อย่างของคำสั่งล่าสุด และฟิลด์ควบคุม 5 อย่างที่ใช้อีเนเบิ้ลอินเตอร์รัพต์หรือควบคุมคุณสมบัติอื่น ๆ
เซกเมนต์รีจิสเตอร์	6:CS,SS,DS,ES, FS,GS	CS ใช้บ่งเซกเมนต์ปัจจุบันสำหรับโค้ด SS ใช้บ่งเซกเมนต์ปัจจุบันสำหรับการเข้าถึงสแต็ก เช่น POP และ PUSH DS ใช้บ่งเซกเมนต์ปัจจุบันของการปฏิบัติการข้อมูลส่วนใหญ่ ES,FS และ GS หมายถึงดาต้าเซกเมนต์อื่น ๆ สำหรับการปฏิบัติการข้อมูล





Status Fields	Present in FLAG (i.e., new with 80386)	Present in EFLAG and FLAG
SF (Sing)	Yes	
PF (Parity)	Yes	
OF (Overflow)	Yes	
ZF (Zero)	Yes	
CF (Carry)	Yes	
AF (Auxiliary Carry)	Yes	
IOPL (I/O Privilege Level)	Yes	
NT (Nested Task)	Yes	
Control Fields		
TF (Trap Flag)	Yes	
IF (Interrupt Enable)	Yes	
DF (Direction Flag)	Yes	
RF (Resume Flag)		Yes
VM (Virtual 8086 Mode)		Yes

รูปที่ 16 EFLAG ของ 80386

EFLAG จะถูกเซตเมื่อผล (result) มีการทด (carry) หรือการยืม (borrow)

สำหรับผู้จะเป็นโปรแกรมเมอร์จะใช้รีจิสเตอร์สำหรับผู้ใช้ (User Register) 16 ตัวนี้เป็นหลัก สำหรับรีจิสเตอร์สำหรับระบบ (system register) จะไม่ขอกล่าวรายละเอียดในที่นี้ จะกล่าวอีกที เมื่อมีบทความกล่าวพาดพิงกับรีจิสเตอร์สำหรับระบบนี้

การอ้างแอดเดรส (addressing mode)

การคำนวณตำแหน่งของแอดเดรสยังผล (effective address: EA) จะคำนวณจากส่วนต่าง ๆ 4 ส่วน ได้แก่ Displacement, Base, Index และ scale ซึ่งจะได้ดังรูปที่ 17 จะมีสมการเป็น

EA = Base Reg + (Index* Scaling) + Displacement

ส่วนต่าง ๆ 4 ส่วน นี้ไม่จำเป็นต้องมีทุกส่วน ซึ่งแบ่งการใช้ส่วนต่าง ๆ 4 ส่วนนี้เป็น addressing mode 9 mode คือ

1. Immediate addressing

เป็นการให้ข้อมูลโดยตรงกับรีจิสเตอร์

```
MOV EAX, OFEED 1234H
MOV EDX, OFFSET mydata
+ 50H
```

2. Register addressing

เป็นการให้ข้อมูลจากรีจิสเตอร์ค้างอื่นกับรีจิสเตอร์ที่เขาต้องการ เช่น

```
MOV EAX, ECX
```

3. Direct memory addressing

เป็นการให้ข้อมูลจากรีจิสเตอร์ตัวอื่นกับรีจิสเตอร์ที่เขาต้องการ เช่น

```
MOV EAX, myvar
```

จะนำข้อมูลจากหน่วยความจำที่ myvar มาเก็บที่ EAX

4. Register indirect addressing

เป็นการให้ข้อมูลจากหน่วยความจำซึ่งใช้รีจิสเตอร์ชี้ที่หน่วยความจำนั้นอีกที่หนึ่ง เช่น

```
LEA BX, MYDATA
MOV AX, [BX]
```

หรือ

```
MOV BX, OFFSET MYDATA
MOV AX, [BX]
```

รีจิสเตอร์ที่ใช้ชี้หน่วยความจำก็ได้แก่ SI, DI, BX และ BP

5. Base Relative addressing

เป็นการอ้างหน่วยความจำโดยใช้เบสรีจิสเตอร์ (E)BX หรือ (E)BP บวกกับดิสเพลคเมนต์ ซึ่งดิสค์เพลคเมนต์ซึ่งมีขนาด 8, 16 หรือ 32 บิต การอ้างแอดเดรสโหมดนี้นิยมใช้กับโครงสร้างข้อมูลที่ซับซ้อน (complex data structure) เช่น ข้อมูลแบบเรคอร์ด เราจะให้เบสรีจิสเตอร์เป็นตัวเลือกเรคอร์ด และใช้ดิสเพลคเมนต์เป็นตัวเลือกข้อมูลในเรคอร์ดนั้น เช่น

```
LEA EBX, MYDATA
MOV EBX, [EBX+4]
```

6. Direct indexed addressing

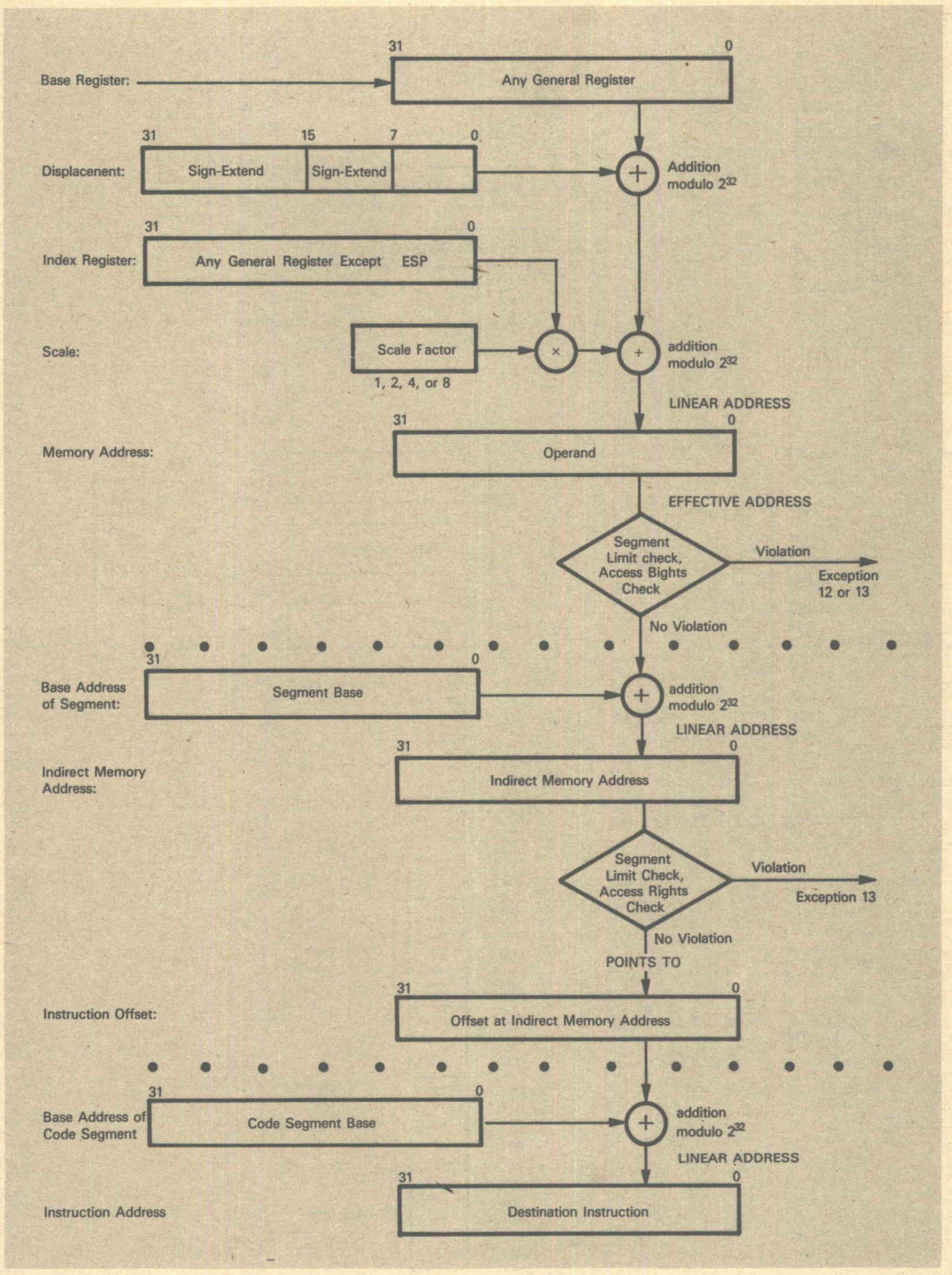
ค่าของออปเซตจะคำนวณจากดิสเพลคเมนต์กับอินเด็กซ์รีจิสเตอร์ (E)SI หรือ E(DI) โหมดนี้จะใช้ในการเข้าถึงข้อมูลแบบสเตติกอะเรย์ โดยดิสเพลคเมนต์จะชี้ที่จุดเริ่มของอะเรย์ และค่าอินเด็กซ์รีจิสเตอร์จะเป็นตัวชี้ข้อมูลในโครงสร้างนั้น เช่น

```
MOV ESI ,4
MOV AX, ARRAY2 [ESI]
```

จะเป็นการนำข้อมูลอะเรย์ 2 ตำแหน่งที่ 4 มาไว้ในรีจิสเตอร์ AX

7. Base indexed addressing

ค่าของออปเซตจะคำนวณจากดิสเพลคเมนต์ อินเด็กซ์รีจิสเตอร์ และเบสรีจิสเตอร์ ใช้กับไดนามิคอะเรย์-



รูปที่ 17
การคำนวณแอดเดรสของหน่วยความจำ

(อะเรย์ที่เบสรีจิสเตอร์เปลี่ยนไปขณะที่โปรแกรมทำงาน) หรือข้อมูลที่เป็นเรคอร์ดและมีข้อมูลข้างในเป็นอะเรย์ เช่น

```
MOV ECX, MYVALUE [BX]
[SI]
```

8. Scaling

มีไว้ใช้ประโยชน์ในการเข้าถึงข้อมูลแบบอะเรย์และข้อมูลแบบโครงสร้างโดยนำอินเด็กซ์รีจิสเตอร์มาคูณกับสเกลแฟคเตอร์ ซึ่งมีค่า 1, 2, 4 หรือ 8 ตัวอย่างเช่น

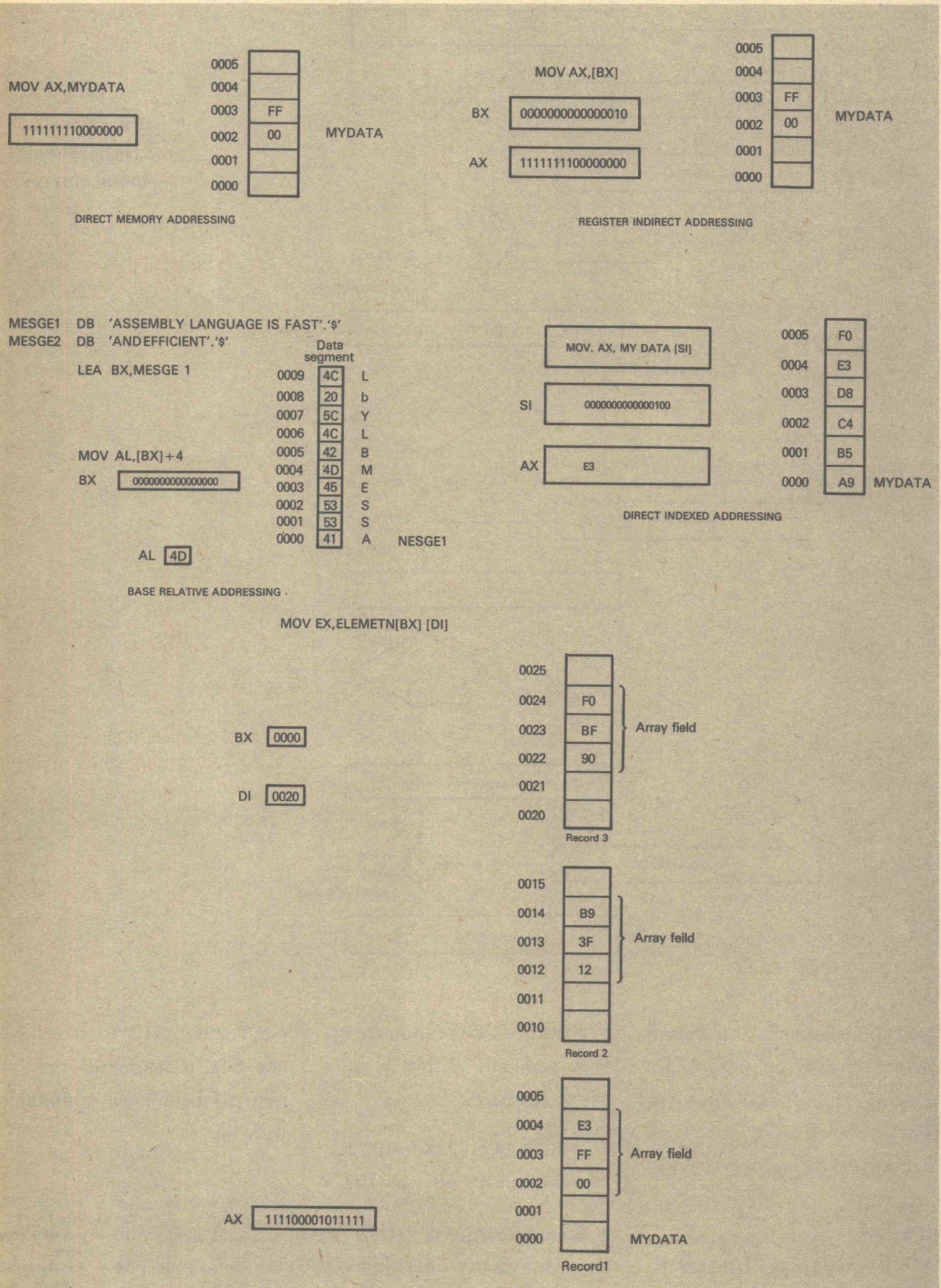
```
MOV EAX, [EBX] [EDX*2]
IMUL EBX, TABLE [ESI*4] , 7
```

9. Bit addressing

เป็นคำสั่งในการทำงานเกี่ยวกับบิตสตริง ซึ่งได้แก่ BT (Test Bit), BTS (Set Test Bit), BTR (Reset Test Bit) และ BTC (Complement Test Bit) เพื่อการเข้าถึงบิตที่ต้องการให้บิตสตริงนั้น ๆ เช่น

```
BT
BT   MEM_WRD OBH ; OBH = immediate offset
BTS  AX CL       ; CL  = offset
BTS  INFO[B J, 7H ; 7H = immediate offset
BTR  AL, AL      ; AL  = offset
BTR  TABLE[S1] OEH ; OEH = immediate offset
BTC  BL, AL      ; AL  = offset
BTC  BUFF[BX], 2H ; 2H = immediate offset
```

รูปที่ 18 การอ้างแอดเดรสแบบต่าง ๆ



การอ้างแอดเดรสทั้ง 9 โหมด นี้ไม่ได้ตายตัว หนังสือเล่มอื่นอาจมีโหมดต่าง ๆ กันออกไปสำหรับโหมดที่ผมนำมาเขียนนี้มาจากหนังสือ Inside The Model 80 โดย Chris H. Happas และ William H. Murrey III ซึ่งลักษณะการอ้างอิงหน่วยความจำพื้นฐานจะไม่ต่างกัน

Flag Word	UUUU0002H Note 1
Machine Status Word (CR0)	UUUUUUU0H Note 2
Instruction Pointer	0000FFF0H
Code Segment	F000H Note 3
Data Segment	0000H
Stack Segment	0000H
Extra Segment (ES)	0000H
Extra Segment (FS)	0000H
Extra Segment (GS)	0000H
DX register	component and stepping ID Note 5
All other registers	undefined Note 4

ตารางที่ 2 แสดงค่าของรีจิสเตอร์หลังจากรีเซต

ชุดคำสั่ง

คำสั่งของ 80386 จะมีรูปแบบดังรูปที่ 19

สำหรับ prefix นั้นใส่หน้าคำสั่งเพื่อให้ได้เงื่อนไขของคำสั่งมีอยู่ 5 ชนิด คือ

- Operand Size ไว้เปลี่ยนขนาดของ operand
- Address Size ไว้เปลี่ยนขนาดของแอดเดรส
- Segment Override เปลี่ยนการเลือก (choice) ของเซกเมนต์
- Bus Lock ให้สัญญาณ bus LOCK #
- Repeat ทำการกระทำซ้ำในคำสั่งที่เกี่ยวกับสตริง จำนวนการทำซ้ำจะอยู่ในรีจิสเตอร์ ECX

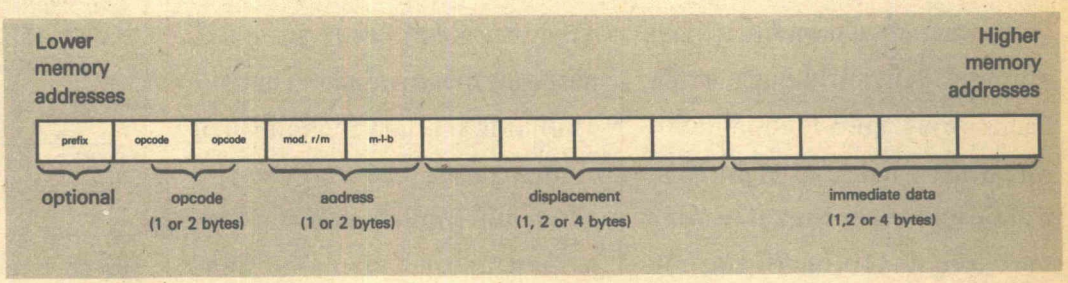
ส่วนคำสั่งนั้นแบ่งได้เป็นกลุ่ม ๆ ได้แก่

1. คำสั่งเกี่ยวกับการเคลื่อนย้ายข้อมูล (Data Movement Instructions)
2. คำสั่งเกี่ยวกับการคำนวณเลขจำนวนเต็ม (Integer Arithmatic Instructions)
3. คำสั่งเกี่ยวกับการกระทำทางลอจิก (Boolean Logical Instructions)
4. คำสั่งเกี่ยวกับการหมุนและเลื่อนข้อมูล (Rotate, Shift และ Double Shift Instructions)
5. คำสั่งเกี่ยวกับการกระทำกับบิต (Bit Manipulation Instructions)
6. คำสั่งเกี่ยวกับการประมวลผลข้อมูลแบบสตริง (String Processing Instructions)
7. คำสั่งเกี่ยวกับไบนารีโค้ดเดซิมอล (Binary Coded Decimal : BCD Arithmatic Instructions)
8. คำสั่งเกี่ยวกับการควบคุมการทำงานของโปรแกรม (Program Control Instructions)
9. คำสั่งเกี่ยวกับการควบคุมระบบ (System Control Instructions)
10. คำสั่งเกี่ยวกับการทำงานให้ซิงโครไนซ์กับโปรเซสเซอร์ช่วย (Coprocessor Data Synchronization Instructions)

การรีเซตและการอินนิเชียไลซ์

เมื่อตัวโปรเซสเซอร์ถูกอินนิเชียไลซ์ หรือรีเซตรีจิสเตอร์ต่าง ๆ จะมีค่าดังแสดงในตารางที่ 2 80386 จะเริ่มการเอ็กซีคิวต์ คำสั่งใกล้ ๆ กับส่วนบนของฟิสิคัลเมมโมรีที่ตำแหน่ง FFFFFFF0H เมื่อมีคำสั่ง JUMP หรือ CALL ระหว่างเซกเมนต์เกิดขึ้นครั้งแรก แอดเดรสบัส A20-A31 สำหรับไซเคิลของหน่วยความจำที่สัมพันธ์กับโค้ดเซกเมนต์จะตกลงเป็น low และ 80386 ก็จะเอ็กซีคิวต์คำสั่งที่ในบริเวณ 1 เมกะไบต์ล่างของฟิสิคัลเมมโมรีทำให้นักออกแบบระบบสามารถใช้ ROM ที่ส่วนบนของหน่วยความจำเพื่ออินนิเชียไลซ์ระบบได้

สัญญาณรีเซตจะบังคับให้ 80386 หยุดการเอ็กซีคิวต์และการแอดดีฟของบัส ระหว่างที่มีสัญญาณรีเซต จะไม่มีการเอ็กซีคิวต์คำสั่งหรือการแอดดีฟของบัส และหลังจากไม่มีสัญญาณรีเซตประมาณ 350 ถึง 450 คาบของสัญญาณนาฬิกา 80386 จะเริ่มเอ็กซีคิวต์คำสั่งที่ส่วนบนของหน่วยความจำ



รูปที่ 19 รูปแบบของคำสั่ง

อินเตอร์รัพต์และเอ็กเซพชัน

อินเตอร์รัพต์และเอ็กเซพชันเป็นการควบคุมชนิดพิเศษ มันจะทำงานเหมือนกับการ CALL ที่ไม่ต้องโปรแกรม

- อินเตอร์รัพต์ เป็นเหตุการณ์ที่เกิดขึ้นไม่พร้อมกัน จะเกิดขึ้นได้โดยถูกทริกจากอุปกรณ์ภายนอก เพื่อให้ซีพียูเปลี่ยนจากทำงานหลักมาทำงานที่อุปกรณ์ภายนอกต้องการให้ทำ

- เอ็กเซพชัน เป็นเหตุการณ์ที่เกิดขึ้นพร้อมกันซึ่งเป็นการตอบรับของซีพียูกับเงื่อนไขที่แน่นอน ซึ่งตรวจสอบเจอระหว่างการเอ็กซีคิวต์คำสั่ง

80386 มีอินเตอร์รัพต์ภายนอก 2 แบบ และเอ็กเซพชัน 2 แบบ

อินเตอร์รัพต์

- Maskable Interrupt เป็นสัญญาณที่ถูกส่งผ่านขา INTR
- Nonmaskable Interrupt เป็นสัญญาณที่ถูกส่งผ่านขา NMI

เอ็กเซพชัน

- ถูกตรวจสอบโดยโปรเซสเซอร์ ซึ่งถูกแบ่งออกเป็น fault, traps, และ aborts
- เกิดจากการโปรแกรม คำสั่ง INT 0, INT 3, INT n และ BOUND สามารถทริกเอ็กเซพชันคำสั่งเหล่านี้มักจะถูกเรียกว่า ซอฟต์แวร์อินเตอร์รัพต์ และตัวโปรเซสเซอร์มันจะจัดการกับมันในฐานะของเอ็กเซพชัน

Faults เป็นเอ็กเซพชันซึ่งถูกรายงานก่อนคำสั่งที่จะทำให้เกิดเอ็กเซพชัน Faults อาจจะถูกตรวจสอบก่อนเอ็กซีคิวต์คำสั่ง หรือระหว่างที่เอ็กซีคิวต์คำสั่ง ถ้าถูกตรวจสอบระหว่างทำคำสั่ง fault จะถูกรายงานโดยซีพียูและกลับไปยังขั้นตอนที่ยอมให้คำสั่งถูกเริ่มต้นใหม่

Traps เป็นเอ็กเซพชันซึ่งถูกรายงานที่ขอบเขตของคำสั่งทันทีทันใดหลังจากคำสั่งที่ซึ่งเอ็กเซพชันถูกตรวจสอบ

Aborts เป็นเอ็กเซพชันถูกใช้เพื่อรายงานข้อผิดพลาดซึ่งร้ายแรง เช่น ข้อผิดพลาดทางฮาร์ดแวร์ หรือค่าที่ไม่ถูกต้องในตารางของระบบ

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	YES	FAULT
Debug Exception	1	any instruction	YES	TRAP*
NMI Interrupt	2	INT 2 or NMI	NO	NMI
One Byte Interrupt	3	INT	NO	TRAP
Interrupt on Overflow	4	INTO	NO	TRAP
Array Bounds Check	5	BOUND	YES	FAULT
Invalid OP-Code	6	Any Illegal Instruction	YES	FAULT
Device Not Available	7	ESC, WAIT	YES	FAULT
Double Fault	8	Any Instruction That Can Generate an Exception		ABORT
Coprocessor Segment Overrun	9	ESC	NO	ABORT
Invalid TSS	10	JMP, CALL, IRET, INT	YES	FAULT
Segment Not Present	11	Segment Register Instructions	YES	FAULT
Stack Fault	12	Stack References	YES	FAULT
General Protection Fault	13	Any Memory Reference	YES	FAULT
Page Fault	14	Any Memory Access or Code Fetch	YES	FAULT
Coprocessor Error	16	ESC, WAIT	YES	FAULT
Intel Reserved	17-32			
Two Byte Interrupt	0-225	INT n	NO	TRAP

ตารางที่ 3 การกำหนดอินเตอร์รัพต์เวกเตอร์

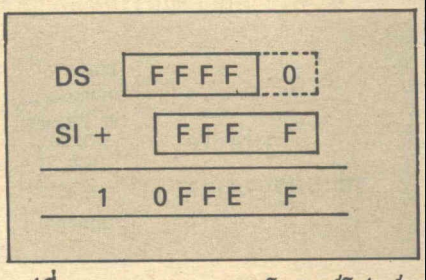
การจัดการกับหน่วยความจำและการอ้างแอดเดรส

หน่วยความจำแบบเซกเมนต์

เช่นเดียวกับไมโครโปรเซสเซอร์ตระกูล 8086 ตัวก่อนหน้า 80386 ก็ใช้สถาปัตยกรรมของการติดต่อหน่วยความจำเป็นแบบเซกเมนต์ ซึ่งมีข้อดีสำหรับระบบขนาดใหญ่ สำหรับผู้ที่เคยเขียนโปรแกรมที่ใช้กับซีพียู 8086 หรือ 8088 อาจจะสงสัยว่าอะไรจะเกิดขึ้นถ้าเบสเซกเมนต์มีค่าใหญ่มาก เช่นเกือบจะถึงตำแหน่ง 1 เมกะไบต์ และค่าออปเซตก็มีค่าใหญ่ด้วย เนื่องจาก 8086 จะอ้างแอดเดรสได้ต้องประกอบด้วย 2 ส่วน เราจะเห็นว่าการอ้างแอดเดรสที่เกินกว่า FFFF สามารถทำได้โดยง่ายดังแสดงในรูปที่ 20

ตัวโปรเซสเซอร์จะจัดการตำแหน่งของแอดเดรสนี้เองโดยการวนขึ้นต้นที่แอดเดรส 00000H ใหม่เมื่อสิ้นสุดแอดเดรสเดิม ซึ่งทำให้เกิดการผิดพลาดได้เพราะว่าการรันโปรแกรมไม่ต่อเนื่องกัน

ในกรณีของ 8086 ออปเซตแอดเดรสอาจจะเป็นค่าคงที่เพื่ออ้างตำแหน่ง



รูปที่ 20 แสดงแอดเดรสโอเวอร์โฟลว์

	EAX	EBX	ECX	EDX	ESI	EDI	EBP	ESP
General storage	✓	✓	✓	✓	✓	✓	✓	*
String operations	✓							
Loop counter			✓					
I/O addresses				✓				
Multiply	✓	✓	✓	✓	✓	✓	✓	✓
Divide (dividend)	✓							
Divide (remainder)				✓				
Base register	✓	✓	✓	✓	✓	✓	✓	✓
Index register	✓	✓	✓	✓	✓	✓	✓	✓
XLAT pointer		✓						
I/O data	✓							
String source					✓			
String destination						✓		

*Not recommended

รูปที่ 21 แสดงการใช้รีจิสเตอร์

ของแอดเดรสในโปรแกรม (เช่น MOV AL,DS:[1547] หรือไม่ก็เป็นการอ้างแอดเดรสทางอ้อมโดยผ่านรีจิสเตอร์ (เช่น MOV AL, DS:[BX] ถ้าเราต้องการใช้รีจิสเตอร์เป็นตัวชี้หน่วยความจำ เราต้องใช้รีจิสเตอร์ตัวใดตัวหนึ่งในสี่ตัว BX และ BP อาจจะใช้เป็นเบสรีจิสเตอร์ ขณะที่ SI และ DI ถูกใช้เป็นอินเด็กซ์-รีจิสเตอร์ คำสั่งเกือบทั้งหมดยอมให้เราใช้ได้ทั้งเบสรีจิสเตอร์และอินเด็กซ์-รีจิสเตอร์ ออปเซตแอดเดรสจะสามารถคำนวณจากผลรวมของค่าภายในรีจิส-เตอร์ทั้งสอง หลักการนี้ยังคงใช้ได้ในไมโครโปรเซสเซอร์ตระกูลนี้จนถึงตัวปัจจุบันคือ 80386

ขณะที่แนวความคิดเหมือนกันนี้ 80386 มีการจัดการกับขีดจำกัดสำหรับรีจิสเตอร์ที่ใช้ในการอ้างแอดเดรส ดูรูปที่ 21 ซึ่งแสดงให้เห็นว่ารีจิสเตอร์ 32 บิตใด ๆ ก็ตามสามารถใช้เป็นเบสรีจิสเตอร์และรีจิสเตอร์ 32 บิตใด ๆ ยกเว้น ESP สามารถใช้เป็นอินเด็กซ์รีจิสเตอร์ได้ ทำให้เกิดความสะดวกมากขึ้นในการอ้างแอดเดรสของหน่วยความจำและยอมให้เราอ้างแอดเดรสที่ก่อนหน้านี้ทำได้ยาก ลองนึกดูว่าถ้าเราใช้ ECX เป็นตัวชี้แอดเดรสจากตำแหน่งหนึ่งที่ใช้ในโครง-สร้างของลูป เราอาจจะคิดว่าควรจะมีการอ้างแอดเดรสแบบอินเด็กซ์ที่เป็นสัด-ส่วน (scaled indexing) ถ้าเราต้องการเช่นนั้นค่าภายในของอินเด็กซ์รีจิสเตอร์สามารถถูกคูณด้วย 2, 4 หรือ 8 ก่อนที่จะใช้เพื่อคำนวณแอดเดรส ซึ่งจะง่ายกว่าการอ้างแบบอะเรย์มาก รูปที่ 22 แสดงคำสั่งที่สามารถอ้างถึงโอเปอร์รันตำแหน่งต่าง ๆ ได้ ซึ่ง 3 คำสั่งสุดท้ายเป็นคุณลักษณะและความสามารถใหม่ของ 80386 ในการอ้างแอดเดรส

จุดสุดท้ายที่มีความสำคัญก็คือ 80386 มีเพจจิ้ง(paging) mmu (memory management unit) ซึ่งสามารถแปลงแอดเดรสก่อนที่จะถูกใช้เป็นฟิสิคัล-แอดเดรส (แอดเดรสที่อ้างถึงตำแหน่งของหน่วยความจำจริง) ซึ่งหมายความว่ามีขบวนการคำนวณแอดเดรสในระดับอื่นรวมทั้งการคำนวณเซกเมนต์และออปเซต ซึ่งออปเซตในที่นี้จะเป็นลอจิคัลแอดเดรส เมื่อลอจิคัลแอดเดรสถูกรวมกับเซกเมนต์แอดเดรสผลที่ได้จะเป็นแอดเดรสแบบเชิงเส้น แอดเดรสเชิงเส้นนี้ต้องผ่าน mmu ก่อนจึงจะได้แอดเดรสที่บัสภายนอกจากของซีพียู ถ้าไม่มีการสั่งให้ทำเพจจิ้ง แอดเดรสเชิงเส้นและแอดเดรสภายนอก (ฟิสิคัลแอดเดรส) ก็จะเป็นแอดเดรสเดียวกัน เช่นเดียวกับ 8086 และ 80286

หน่วยความจำแคช (Cache memory)

ถ้าเราให้ 80386 ทำงานโดยใช้สัญญาณนาฬิกาที่มีความถี่ 16 เมกะ-เฮิรตซ์ มันจะสามารถจะจบไซเคิลของบัสได้ในเวลาเพียง 125 นาโนวินาที สำหรับแบนวิดซ์ที่สูงที่สุดของ 32 เมกะ-ไบต์ต่อวินาทีเพื่อที่จะให้ 80386 ทำงานด้วยความเร็วสูงสุดนี้ เราจำเป็นต้องใช้หน่วยความจำที่มีคุณภาพสูง คือมี access time น้อย ๆ เพื่อที่จะสามารถจบไซเคิลของบัสได้โดยไม่มี wait state และหน่วยความจำต้องมีขนาดใหญ่พอที่จะให้ 80386 เอ็กซีคิวต์โปรแกรมขนาดใหญ่ได้

โดยปกติแล้วหน่วยความจำจะเป็นชนิดไดนามิคแรม (DRAM) ซึ่งมีความหนาแน่นสูง (คือ มีจำนวนที่เก็บข้อมูลมากแต่ตัวเล็ก) และราคาถูก เพื่อประ-หยัดพื้นที่และประหยัดเงิน อย่างไรก็ตาม

Mode	Example
Immediate constant	MOV EAX, 12345678
Immediate register	MOV EAX, ECX
Direct (offset only)	MOV EAX, [12345678]
Register indirect	MOV EAX, [ECX]
Base/Index + offset	MOV EAX, [ECX] + 9930
Base – index + offset	MOV EAX, [ECX][EDX] + 42
Index * scale + offset	MOV EAX, [ESI*4] + 917
Base – (index*scale)	MOV EAX, [EDX][ECX*8]
Base – (index*scale) + offset	MOV EAX, [EBX][EDI*2] + 316

รูปที่ 22 แอดเดรสซิ่งโหมด

ไม่มี DRAM ธรรมดาใด ๆ ที่มีราคาถูกและสามารถทำงานให้จบไซเคิลได้ในเวลา 125 นาโนวินาที static RAM (SRAM) ซึ่งเร็วกว่า สามารถทำงานนี้ได้ แต่มีข้อเสียคือ ความแน่นอนต่ำกว่า และมีราคาสูงมาก ซึ่งไม่เหมาะที่จะนำมาใช้งาน

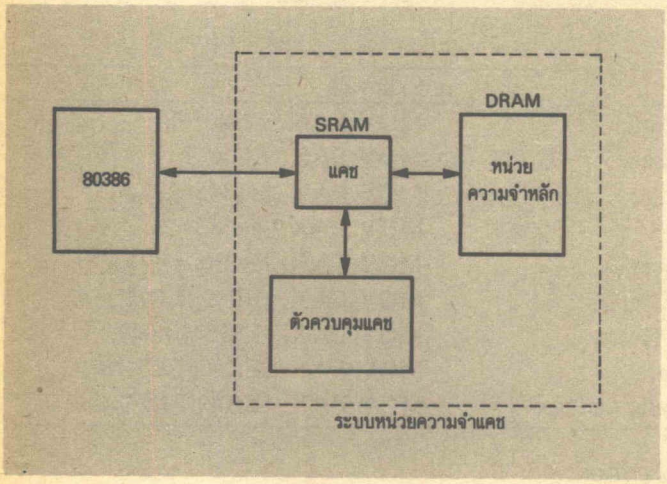
ระบบหน่วยความจำแบบแคช จะเป็นวิธีการจัดการหน่วยความจำที่สามารถแก้ปัญหาดังกล่าวได้ โดยระบบแคชจะประกอบด้วยหน่วยความจำที่มีความเร็วสูงซึ่งเป็น static RAM จำนวนน้อยอยู่ส่วนหนึ่ง และหน่วยความจำที่มีความเร็วน้อยกว่าซึ่งเป็น dynamic RAM อยู่จำนวนมาก ระบบหน่วยความจำจะเสมือนถูกจำลองเป็นหน่วยความจำที่มีความเร็วสูงจำนวนมาก ดังนั้นระบบแคชจึงมีความสามารถและคุณสมบัติของ static RAM แต่มีราคาอยู่ในระดับของ dynamic RAM

ระบบหน่วยความจำแคช แสดงดังรูปที่ 23 ซึ่งประกอบด้วยส่วนต่าง ๆ ดังนี้

- แคช (cache) ซึ่งเป็น static RAM ความเร็วสูงระหว่างตัวโปรเซสเซอร์กับหน่วยความจำหลักที่มีความเร็วต่ำกว่า
- หน่วยความจำหลักแบบ DRAM
- ตัวควบคุมแคช (cache controller)

ในระบบหน่วยความจำแคช ข้อมูลทั้งหมดจะถูกเก็บอยู่ในหน่วยความจำหลัก และข้อมูลบางส่วนจะถูกก๊อบปี้ไปเก็บในแคช เมื่อตัวโปรเซสเซอร์แอคเซสกับหน่วยความจำ มันจะเช็คดูที่แคชก่อน ถ้าข้อมูลที่ต้องการอยู่ในแคชมันก็จะแอคเซสข้อมูลได้เร็ว แต่ถ้าข้อมูลที่ต้องการไม่อยู่ในแคชมันจะต้องไปเฟตช์ (fetch) มาจากหน่วยความจำหลัก

แคชสามารถลดเวลาโดยเฉลี่ยของการแอคเซสหน่วยความจำได้ ถ้าสามารถจัดการให้โค้ดและข้อมูลที่ต้องการแอคเซสบ่อย ๆ อยู่ในแคช โปรแกรมจะสามารถทำงานได้เร็วขึ้นถ้าการกระทำส่วนใหญ่ติดต่อกับแคช โอกาสที่ตัวโปรเซสเซอร์จะพบข้อมูลที่ต้องการในแคชนั้นขึ้นกับปัจจัยต่าง ๆ เช่น ขนาดของแคช การจัดองค์ประกอบของแคช อัลกอริทึมของแคช และโปรแกรมที่กำลังถูกรันโดยทั่วไปแล้วโปรแกรมมักจะแอคเซสหน่วยความจำในบริเวณใกล้เคียงกับที่เพิ่งจะแอคเซสไปก่อนหน้า เช่น การเอ็กซีคิวต์คำสั่งมักจะทำเป็นลำดับหรือในลูปเล็ก ๆ การแอคเซสหน่วยความจำจึงอยู่ในบริเวณใกล้เคียงกัน ตัวแปรต่าง ๆ ก็จะถูกอ้างถึงบ่อยครั้งในการทำงานครั้งหนึ่ง ๆ ส่วนการเพิ่มขึ้นหรือลดลงของสแต็กก็เกิดขึ้นจากด้านใดด้านหนึ่ง ทำให้การแอคเซสในครั้งต่อ ๆ ไปก็จะอยู่บริเวณส่วนบนของสแต็ก



รูปที่ 23 ระบบหน่วยความจำแคช

จากหลักการดังกล่าวนี้ทำให้ระบบหน่วยความจำแคชทำงานได้ดี แต่ก็ไม่เสมอไปสำหรับโปรแกรมทุกโปรแกรม การกระโดดในโปรแกรมและการสลับไปมาของโปรแกรมเป็นตัวอย่างที่ทำให้ระบบหน่วยความจำแคชไม่เกิดประโยชน์ขึ้นเลย

ทั้งหมดที่กล่าวมานี้เป็นข้อเด่นของซีพียูเบอร์ 80386 และเป็นความสามารถของเครื่องไมโครคอมพิวเตอร์ที่มี 80386 เป็นตัวประมวลผลกลางอย่างย่อ ๆ ส่วนในรายละเอียดของความสามารถด้านต่าง ๆ เช่น หน่วยความจำแบบเซกเมนต์ การทำงานในโหมดต่าง ๆ การจัดการหน่วยความจำแบบเพจจิ้ง ระบบหน่วยความจำแบบแคช (cache memory) จอภาพระบบ VGA และอื่น ๆ จะนำมาเสนอต่อไป

หมายเหตุ

หนังสือที่ใช้อ่านเพิ่มเติม

1. 80386 Hardware Referance Manual ของอินเทล เป็นเล่มที่อธิบายเกี่ยวกับฮาร์ดแวร์ของ 80386

2. 80386 Programmer's Reference Manual ของอินเทล เป็นหนังสือที่อ้างอิงในการเขียนโปรแกรม

3. Advanced 80386 Programming Techniques ของ James L. Turley สำนักพิมพ์ McGraw-Hill เป็นหนังสือที่แสดงการเขียนโปรแกรมภายใต้ 80386 ซึ่งเป็นเล่มที่ดีมากเล่มหนึ่ง

4. 80386 Technical Reference ของ Edmund Straus สำนักพิมพ์ Prentice Hall เล่มนี้เป็นหนังสือที่ผู้แต่งเป็นคนในทีมงานที่ออกแบบ 8086, 80286 และ 80386

5. Dr.Dobf's Toolbook of 80286/80386 Programming โดย Phillip Robinson จะมีการกล่าวถึงการเขียนโปรแกรมของ 80286 80386 และ 82786 (Graphic Coprocessor) การใช้การจัดการหน่วยความจำเพื่อสนับสนุน OS แบบมัลติทาสกิ้ง และมัลติยูสเซอร์ การ optimize การทำงานทางด้านกราฟิกและ floating point